

Optimization of Lyapunov Invariants in Verification of Software Systems

Mardavij Roozbehani, *Member, IEEE*, Alexandre Megretski, *Member, IEEE*, and Eric Feron *Member, IEEE*

Abstract

The paper proposes a control-theoretic framework for verification of numerical software systems, and puts forward software verification as an important application of control and systems theory. The idea is to transfer Lyapunov functions and the associated computational techniques from control systems analysis and convex optimization to verification of various software safety and performance specifications. These include but are not limited to absence of overflow, absence of division-by-zero, termination in finite time, presence of dead-code, and certain user-specified assertions. Central to this framework are Lyapunov invariants. These are properly constructed functions of the program variables, and satisfy certain properties—resembling those of Lyapunov functions—along the execution trace. The search for the invariants can be formulated as a convex optimization problem. If the associated optimization problem is feasible, the result is a certificate for the specification.

Index Terms

Software Verification, Lyapunov Invariants, Convex Optimization.

I. INTRODUCTION

SFTWARE in safety-critical systems implement complex algorithms and feedback laws that control the interaction of physical devices with their environments. Examples of such systems are abundant in aerospace, automotive, and medical applications. The range of theoretical and practical issues that arise in analysis, design, and implementation of safety-critical software systems is extensive, see, e.g., [31], [44], and [27]. While safety-critical software must satisfy various resource allocation, timing, scheduling, and fault tolerance constraints, the foremost requirement is that it must be free of run-time errors.

A. Overview of Existing Methods

1) *Formal Methods*: Formal verification methods are model-based techniques [51], [48], [41] for proving or disproving that a mathematical model of a software (or hardware) satisfies a given *specification*,

i.e., a mathematical expression of a desired behavior. The approach adopted in this paper too, falls under this category. Herein, we briefly review *model checking* and *abstract interpretation*.

a) Model Checking: In *model checking* [15] the system is modeled as a finite state transition system and the specifications are expressed in some form of logic formulae, e.g., temporal or propositional logic. The verification problem then reduces to a graph search, and symbolic algorithms are used to perform an exhaustive exploration of all possible states. Model checking has proven to be a powerful technique for verification of circuits [14], security and communication protocols [38], [45] and stochastic processes [4]. Nevertheless, when the program has non-integer variables, or when the state space is continuous, model checking is not directly applicable. In such cases, combinations of various abstraction techniques and model checking have been proposed [3], [19], [62]; scalability, however, remains a challenge.

b) Abstract Interpretation: is a theory for formal approximation of the *operational semantics* of computer programs in a systematic way [16]. Construction of abstract models involves abstraction of domains—typically in the form of a combination of sign, interval, polyhedral, and congruence abstractions of sets of data—and functions. A system of fixed-point equations is then generated by symbolic forward/backward executions of the abstract model. An iterative equation solving procedure, e.g., Newton’s method, is used for solving the nonlinear system of equations, the solution of which results in an inductive invariant assertion, which is then used for checking the specifications. In practice, to guarantee finite convergence of the iterates, narrowing (outer approximation) operators are constructed to estimate the solution, followed by widening (inner approximation) to improve the estimate [17]. This compromise can be a source of conservatism in analysis [1]. Nevertheless, these methods have been used in practice for verification of limited properties of embedded software of commercial aircraft [8].

Alternative formal methods can be found in the computer science literature mostly under *deductive verification* [37], *type inference* [52], and *data flow analysis* [28]. These methods share extensive similarities in that a notion of program abstraction and symbolic execution or constraint propagation is present in all of them. Further details and discussions of the methodologies can be found in [17], and [48].

2) *System Theoretic Methods*: While software analysis has been the subject of an extensive body of research in computer science, treatment of the topic in the control systems community has been less systematic. The relevant results in the systems and control literature can be found in the field of hybrid systems [12]. Much of the available techniques for safety verification of hybrid systems are explicitly or implicitly based on computation of the reachable sets, either exactly or approximately. These include but are not limited to techniques based on quantifier elimination [34], ellipsoidal calculus [32], and mathematical programming [6]. Alternative approaches aim at establishing properties of hybrid systems through barrier certificates [53], numerical computation of Lyapunov functions [11], [29], or by combined use of bisimulation mechanisms and Lyapunov techniques [24], [33], [62], [3].

Inspired by the concept of Lyapunov functions in stability analysis of nonlinear dynamical systems [30], in this paper we propose Lyapunov invariants for analysis of computer programs. While Lyapunov functions and similar concepts have been used in verification of stability or temporal properties of system level descriptions of hybrid systems [55], [11], [29], to the best of our knowledge, this paper is the first to present a systematic framework based on Lyapunov invariance and convex optimization for verification of a broad range of code-level specifications for computer programs¹. Accordingly, it is in the systematic integration of new ideas and some well-known tools within a unified software analysis framework that we see the main contribution of our work, and not in carrying through the proofs of the underlying theorems and propositions. The introduction and development of such framework provides an opportunity for the field of *control* to systematically address a problem of great practical significance and interest to both computer science and engineering communities. The framework can be summarized as follows:

- 1) Dynamical system interpretation and modeling (Section II). We introduce generic dynamical system representations of programs, along with specific modeling languages which include Mixed-Integer Linear Models (MILM), Graph Models, and MIL-over-Graph Hybrid Models (MIL-GHM).
- 2) Lyapunov invariants as behavior certificates for computer programs (Section III). Analogous to a Lyapunov function, a Lyapunov invariant is a real-valued function of the program variables, and satisfies a *difference inequality* along the trace of the program. It is shown that such functions can

¹This paper constitutes the synthesis and extension of ideas and computational techniques expressed in the workshop [20], and subsequent conference papers [56], [57] and [58]. Some of the ideas presented in [20], [56], and [57] were independently reported in [18].

be formulated for verification of various specifications.

- 3) A computational procedure for finding the Lyapunov invariants (Section IV). The procedure is standard and constitutes these steps: (i) Restricting the search space to a linear subspace. (ii) Using convex relaxation techniques to formulate the search problem as a convex optimization problem, e.g., a Linear Program (LP) [7], Semidefinite Program (SDP) [10], [63], or a Sum-of-Squares (SOS) program [49]. (iii) Using convex optimization software for numerical computation of the certificates.

II. DYNAMICAL SYSTEM INTERPRETATION AND MODELING OF COMPUTER PROGRAMS

We interpret computer programs as discrete-time dynamical systems and introduce generic models that formalize this interpretation. We then introduce MILMs, Graph Models, and MIL-GHMs as structured cases of the generic models. The specific modeling languages are used for computational purposes.

A. Generic Models

1) *Concrete Representation of Computer Programs:* We will consider generic models defined by a finite state space set X with selected subsets $X_0 \subseteq X$ of initial states, and $X_\infty \subset X$ of terminal states, and by a set-valued state transition function $f : X \mapsto 2^X$, such that $f(x) \subseteq X_\infty, \forall x \in X_\infty$. We denote such dynamical systems by $\mathcal{S}(X, f, X_0, X_\infty)$.

Definition 1: The dynamical system $\mathcal{S}(X, f, X_0, X_\infty)$ is a \mathcal{C} -representation of a computer program \mathcal{P} , if the set of all sequences that can be generated by \mathcal{P} is equal to the set of all sequences $\mathcal{X} = (x(0), x(1), \dots, x(t), \dots)$ of elements from X , satisfying

$$x(0) \in X_0 \subseteq X, \quad x(t+1) \in f(x(t)) \quad \forall t \in \mathbb{Z}_+ \quad (1)$$

The uncertainty in $x(0)$ allows for dependence of the program on different initial conditions, and the uncertainty in f models dependence on parameters, as well as the ability to respond to real-time inputs.

Example 1: Integer Division (adopted from [51]): The functionality of Program 1 is to compute the result of the integer division of dd (dividend) by dr (divisor). A \mathcal{C} -representation of the program is displayed alongside. Note that if $dd \geq 0$, and $dr \leq 0$, then the program never exits the “while” loop and the value of q keeps increasing, eventually leading to either an overflow or an erroneous answer. The program terminates if dd and dr are positive.

```

int IntegerDivision ( int dd,int dr )
{int q = {0}; int r = {dd};
while (r >= dr)
{  q = q + 1;
  r = r - dr; }
return r; }

```

```

 $\mathbb{Z} = \mathbb{Z} \cap [-32768, 32767]$ 
 $X = \mathbb{Z}^4$ 
 $X_0 = \{(dd, dr, q, r) \in X \mid q = 0, r = dd\}$ 
 $X_\infty = \{(dd, dr, q, r) \in X \mid r < dr\}$ 
 $f : (dd, dr, q, r) \mapsto \begin{cases} (dd, dr, q + 1, r - dr), & (dd, dr, q, r) \in X \setminus X_\infty \\ (dd, dr, q, r), & (dd, dr, q, r) \in X_\infty \end{cases}$ 

```

Program 1: The Integer Division Program (left) and its Dynamical System Model (right)

2) *Abstract Representation of Computer Programs:* In a \mathcal{C} -representation, the elements of the state space X belong to a finite subset of the set of rational numbers that can be represented by a fixed number of bits in a specific arithmetic framework, e.g., fixed-point or floating-point arithmetic. When the elements of X are non-integers, due to the quantization effects, the set-valued map f often defines very complicated dependencies between the elements of X , even for simple programs involving only elementary arithmetic operations. An abstract model over-approximates the behavior set in the interest of tractability. The drawbacks are conservatism of the analysis and (potentially) undecidability. Nevertheless, abstractions in the form of formal over-approximations make it possible to formulate computationally tractable, sufficient conditions for a verification problem that would otherwise be intractable.

Definition 2: Given a program \mathcal{P} and its \mathcal{C} -representation $\mathcal{S}(X, f, X_0, X_\infty)$, we say that $\bar{\mathcal{S}}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$ is an \mathcal{A} -representation, i.e., an *abstraction* of \mathcal{P} , if $X \subseteq \bar{X}$, $X_0 \subseteq \bar{X}_0$, and $f(x) \subseteq \bar{f}(x)$ for all $x \in X$, and the following condition holds:

$$\bar{X}_\infty \cap X \subseteq X_\infty. \quad (2)$$

Thus, every trajectory of the actual program is also a trajectory of the abstract model. The definition of \bar{X}_∞ is slightly more subtle. For proving Finite-Time Termination (FTT), we need to be able to infer that if all the trajectories of $\bar{\mathcal{S}}$ eventually enter \bar{X}_∞ , then all trajectories of \mathcal{S} will eventually enter X_∞ . It is tempting to require that $\bar{X}_\infty \subseteq X_\infty$, however, this may not be possible as X_∞ is often a discrete set, while \bar{X}_∞ is dense in the domain of real numbers. The definition of \bar{X}_∞ as in (2) resolves this issue.

Construction of $\bar{\mathcal{S}}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$ from $\mathcal{S}(X, f, X_0, X_\infty)$ involves abstraction of each of the elements X, f, X_0, X_∞ in a way that is consistent with Definition 2. Abstraction of the state space X often involves

replacing the domain of *floats* or integers or a combination of these by the domain of real numbers. Abstraction of X_0 or X_∞ often involves a combination of domain abstractions and abstraction of functions that define these sets. Semialgebraic set-valued abstractions of some commonly used nonlinearities is presented in Appendix I. Also, abstractions of fixed-point and floating point arithmetic operations based on ideas from [42] and [43] are discussed in Appendix I. A case study in application of these methods to analysis of a program with floating-point operations is presented in Section V-B.

B. Specific Models of Computer Programs

Specific modeling languages are particularly useful for automating the proof process in a computational framework. Here, three specific modeling languages are proposed: *Mixed-Integer Linear Models (MILM)*, *Graph Models*, and *Mixed-Integer Linear over Graph Hybrid Models (MIL-GHM)*.

1) *Mixed-Integer Linear Model (MILM)*: Proposing MILMs for software modeling and analysis is motivated by the observation that by imposing linear equality constraints on boolean and continuous variables over a quasi-hypercube, one can obtain a relatively compact representation of arbitrary piecewise affine functions defined over compact polytopic subsets of Euclidean spaces (Proposition 1). The earliest reference to the statement of universality of MILMs appears to be [46], in which a constructive proof is given for the one-dimensional case. A constructive proof for the general case is given in Appendix II.

Proposition 1: Universality of Mixed-Integer Linear Models. Let $f : X \mapsto \mathbb{R}^n$ be a piecewise affine map with a closed graph, defined on a compact state space $X \subseteq [-1, 1]^n$, consisting of a finite union of compact polytopes. That is:

$$f(x) \in 2A_i x + 2B_i \quad \text{subject to} \quad x \in X_i, \quad i \in \mathbb{Z}(1, N)$$

where, each X_i is a compact polytopic set. Then, f can be specified precisely, by imposing linear equality constraints on a finite number of binary and continuous variables ranging over compact intervals.

Specifically, there exist matrices F and H , such that the following two sets are equal:

$$G_1 = \{(x, f(x)) \mid x \in X\}$$

$$G_2 = \{(x, y) \mid F[x \ w \ v \ 1]^T = y, \ H[x \ w \ v \ 1]^T = 0, \ (w, v) \in [-1, 1]^{n_w} \times \{-1, 1\}^{n_v}\}$$

Mixed Logical Dynamical Systems (MLDS) with similar structure were considered in [5] for analysis of a class of hybrid systems. The main contribution here is in the application of the model to software analysis. A MIL model of a computer program is defined via the following elements:

- 1) The state space $X \subset [-1, 1]^n$.
- 2) Letting $n_e = n + n_w + n_v + 1$, the state transition function $f : X \mapsto 2^X$ is defined by two matrices F , and H of dimensions n -by- n_e and n_H -by- n_e respectively, according to:

$$f(x) \in \left\{ F \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T \mid H \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, (w, v) \in [-1, 1]^{n_w} \times \{-1, 1\}^{n_v} \right\}. \quad (3)$$

- 3) The set of initial conditions is defined via either of the following:
 - a) If X_0 is finite with a small cardinality, then it can be conveniently specified by extension. We will see in Section IV that per each element of X_0 , one constraint needs to be included in the set of constraints of the optimization problem associated with the verification task.
 - b) If X_0 is not finite, or $|X_0|$ is too large, an abstraction of X_0 can be specified by a matrix $H_0 \in R^{n_{H_0} \times n_e}$ which defines a union of compact polytopes in the following way:

$$X_0 = \{x \in X \mid H_0 \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, (w, v) \in [-1, 1]^{n_w} \times \{-1, 1\}^{n_v}\}. \quad (4)$$

- 4) The set of terminal states X_∞ is defined by

$$X_\infty = \{x \in X \mid H \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T \neq 0, \forall w \in [-1, 1]^{n_w}, \forall v \in \{-1, 1\}^{n_v}\}. \quad (5)$$

Therefore, $\mathcal{S}(X, f, X_0, X_\infty)$ is well defined. A compact description of a MILM of a program is either of the form $\mathcal{S}(F, H, H_0, n, n_w, n_v)$, or of the form $\mathcal{S}(F, H, X_0, n, n_w, n_v)$. The MILMs can represent a broad range of computer programs of interest in control applications, including but not limited to control programs of gain scheduled linear systems in embedded applications. In addition, generalization of the model to programs with piecewise affine dynamics subject to quadratic constraints is straightforward.

Example 2: A MILM of an abstraction of the IntegerDivision program (Program 1: Section II-A), with

all the integer variables replaced with real variables, is given by $\mathcal{S}(F, H, H_0, 4, 3, 0)$, where

$$H_0 = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 1 & 0 & 1 \\ -2 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad H = \begin{bmatrix} 0 & 2 & 0 & -2 & 1 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 & 0 & 1 & 0 & 1 \\ -2 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1/M \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Here, M is a scaling parameter used for bringing all the variables within the interval $[-1, 1]$.

2) *Graph Model*: Practical considerations such as universality and strong resemblance to the natural flow of computer code render graph models an attractive and convenient model for software analysis. Before we proceed, for convenience, we introduce the following notation: $P_r(i, x)$ denotes the projection operator defined as $P_r(i, x) = x$, for all $i \in \mathbb{Z} \cup \{\infty\}$, and all $x \in \mathbb{R}^n$.

A graph model is defined on a directed graph $G(\mathcal{N}, \mathcal{E})$ with the following elements:

- 1) A set of nodes $\mathcal{N} = \{\emptyset\} \cup \{1, \dots, m\} \cup \{\infty\}$. These can be thought of as line numbers or code locations. Nodes \emptyset and ∞ are starting and terminal nodes, respectively. The only possible transition from node ∞ is the identity transition to node ∞ .
- 2) A set of edges $\mathcal{E} = \{(i, j, k) \mid i \in \mathcal{N}, j \in \mathcal{O}(i)\}$, where the *outgoing set* $\mathcal{O}(i)$ is the set of all nodes to which transition from node i is possible in one step. Definition of the *incoming set* $\mathcal{I}(i)$ is analogous. The third element in the triplet (i, j, k) is the index for the k th edge between i and j , and $\mathcal{A}_{ji} = \{k \mid (i, j, k) \in \mathcal{E}\}$.
- 3) A set of program variables $x_l \in \Omega \subseteq \mathbb{R}$, $l \in \mathbb{Z}(1, n)$. Given \mathcal{N} and n , the state space of a graph model is $X = \mathcal{N} \times \Omega^n$. The state $\tilde{x} = (i, x)$ of a graph model has therefore, two components: The discrete component $i \in \mathcal{N}$, and the continuous component $x \in \Omega^n \subseteq \mathbb{R}^n$.
- 4) A set of *transition* labels \bar{T}_{ji}^k assigned to every edge $(i, j, k) \in \mathcal{E}$, where \bar{T}_{ji}^k maps x to the set $\bar{T}_{ji}^k x = \{T_{ji}^k(x, w, v) \mid (x, w, v) \in S_{ji}^k\}$, where $(w, v) \in [-1, 1]^{n_w} \times [-1, 1]^{n_v}$, and $T_{ji}^k : \mathbb{R}^{n+n_w+n_v} \mapsto \mathbb{R}^n$ is a polynomial function and S_{ji}^k is a semialgebraic set. If \bar{T}_{ji}^k is a deterministic map, we drop S_{ji}^k and define $\bar{T}_{ji}^k \equiv T_{ji}^k(x)$.
- 5) A set of *passport* labels Π_{ji}^k assigned to all edges $(i, j, k) \in \mathcal{E}$, where Π_{ji}^k is a semialgebraic set. A state transition along edge (i, j, k) is possible if and only if $x \in \Pi_{ji}^k$.

6) Semialgebraic invariant sets $X_i \subseteq \Omega^n$, $i \in \mathcal{N}$ are assigned to every node on the graph, such that

$P_r(i, x) \in X_i$. Equivalently, a state $\tilde{x} = (i, x)$ satisfying $x \in X \setminus X_i$ is unreachable.

Therefore, a graph model is a well-defined specific case of the generic model $\mathcal{S}(X, f, X_0, X_\infty)$, with $X = \mathcal{N} \times \Omega^n$, $X_0 = \{\emptyset\} \times X_\emptyset$, $X_\infty = \{\bowtie\} \times X_\bowtie$ and $f : X \mapsto 2^X$ defined as:

$$f(\tilde{x}) \equiv f(i, x) = \left\{ (j, \overline{T}_{ji}^k x) \mid j \in \mathcal{O}(i), x \in \Pi_{ji}^k \cap X_i \right\}. \quad (6)$$

Conceptually similar models, namely control flow graphs, have been reported in [51] (and the references therein) for software verification, and in [2], [13] for modeling and verification of hybrid systems.

Remarks

- The invariant set of node \emptyset contains all the available information about the initial conditions of the program variables: $P_r(\emptyset, x) \in X_\emptyset$.

- Multiple edges between nodes enable modeling of logical "or" or "xor" type conditional transitions.

This allows for modeling systems with nondeterministic discrete transitions.

- The transition label \overline{T}_{ji}^k may represent a simple update rule which depends on the real-time input.

For instance, if $T = Ax + Bw$, and $S = \mathbb{R}^n \times [-1, 1]$, then $x \xrightarrow{\overline{T}} \{Ax + Bw \mid w \in [-1, 1]\}$.

In other cases, \overline{T}_{ji}^k may represent an abstraction of a nonlinearity. For instance, the assignment

$x \mapsto \sin(x)$ can be abstracted by $x \xrightarrow{\overline{T}} \{T(x, w) \mid (x, w) \in S\}$ (see Eqn. (63) in Appendix I).

- Graph models with state-dependent or time-varying transitions labels arise, for instance, in modeling computer programs that involve arithmetic operations with array elements. For instance, consider the case where the transition functions are parametrized, and the parameters are drawn from a finite set, e.g., a multidimensional array. The dependence on the array elements can be random, resulting time-varying transition labels, or it can be in the form of a functional dependence on other program variables. Systematic treatment of such models is discussed in Appendix I.

Before we proceed, we introduce the following notation: Given a semialgebraic set Π , and a polynomial function $\tau : \mathbb{R}^n \mapsto \mathbb{R}^n$, we denote by $\Pi(\tau)$, the set: $\Pi(\tau) = \{x \mid \tau(x) \in \Pi\}$.

a) Construction of Simple Invariant Sets: Simple invariant sets can be included in the model if they are readily available or easily computable. Even trivial invariants can simplify the analysis and improve

the chances of finding stronger invariants via convex relaxations, e.g., the \mathcal{S} -Procedure (cf. Section IV).

- Simple invariant sets may be provided by the programmer. These can be trivial sets representing simple algebraic relations between variables, or they can be more complicated relationships that reflect the programmer’s knowledge about the functionality and behavior of the program.
- Invariant Propagation: Assuming that T_{ij}^k are deterministic and invertible, the set

$$X_i = \bigcup_{j \in \mathcal{I}(i), k \in \mathcal{A}_{ij}} \Pi_{ij}^k \left((T_{ij}^k)^{-1} \right) \quad (7)$$

is an invariant set for node i . Furthermore, if the invariant sets X_j are strict subsets of Ω^n for all $j \in \mathcal{I}(i)$, then (7) can be improved. Specifically, the set

$$X_i = \bigcup_{j \in \mathcal{I}(i), k \in \mathcal{A}_{ij}} \Pi_{ij}^k \left((T_{ij}^k)^{-1} \right) \cap X_j \left((T_{ij}^k)^{-1} \right) \quad (8)$$

is an invariant set for node i . Note that it is sufficient that the restriction of T_{ij}^k to the lower dimensional spaces in the domains of Π_{ij}^k and X_j be invertible.

- Preserving Equality Constraints: Simple assignments of the form $T_{ij}^k : x_l \mapsto f(y_m)$ result in invariant sets of the form $X_i = \{x \mid x_l - f(y_m) = 0\}$ at node i , provided that T_{ij}^k does not simultaneously update y_m . Formally, let T_{ij}^k be such that $(T_{ij}^k x)_l - x_l$ is non-zero for at most one element $\hat{l} \in \mathbb{Z}(1, n)$, and that $(T_{ij}^k x)_{\hat{l}}$ is independent of $x_{\hat{l}}$. Then, the following set is an invariant set at node i :

$$X_i = \bigcup_{j \in \mathcal{I}(i), k \in \mathcal{A}_{ij}} \{x \mid [T_{ij}^k - I] x = 0\}$$

3) *Mixed-Integer Linear over Graph Hybrid Model (MIL-GHM)*: The MIL-GHMs are graph models in which the effects of several lines and/or *functions* of code are compactly represented via a MILM. As a result, the graphs in such models have edges (possibly self-edges) that are labeled with matrices F and H corresponding to a MILM as the transition and passport labels. Such models combine the flexibility provided by graph models and the compactness of MILMs. An example is presented in Section V.

C. Specifications

The specification that can be verified in our framework can generically be described as unreachability and finite-time termination.

Definition 3: A Program $\mathcal{P} \equiv \mathcal{S}(X, f, X_0, X_\infty)$ is said to satisfy the unreachability property with respect to a subset $X_- \subset X$, if for every trajectory $\mathcal{X} \equiv x(\cdot)$ of (1), and every $t \in \mathbb{Z}_+$, $x(t)$ does not belong to X_- . A program $\mathcal{P} \equiv \mathcal{S}(X, f, X_0, X_\infty)$ is said to *terminate in finite time* if every solution $\mathcal{X} = x(\cdot)$ of (1) satisfies $x(t) \in X_\infty$ for some $t \in \mathbb{Z}_+$.

Several critical specifications associated with runtime errors are special cases of unreachability.

1) *Overflow:* Absence of overflow can be characterized as a special case of unreachability by defining:

$$X_- = \{x \in X \mid \|\alpha^{-1}x\|_\infty > 1, \alpha = \text{diag}\{\alpha_i\}\}$$

where $\alpha_i > 0$ is the overflow limit for variable i . We say that $\alpha \succ 0$ specifies the overflow limit.

2) *Out-of-Bounds Array Indexing:* An out-of-bounds array indexing error occurs when a variable exceeding the length of an array, references an element of the array. Assuming that x_l is the corresponding integer index and L is the array length, one must verify that x_l does not exceed L at location i , where referencing occurs. This can be accomplished by defining $X_- = \{(i, x) \in X \mid |x_l| > L\}$ over a graph model and proving that X_- is unreachable. This is also similar to “assertion checking” defined next.

3) *Program Assertions:* An *assertion* is a mathematical expression whose validity at a specific location in the code must be verified. It usually indicates the programmer’s expectation from the behavior of the program. We consider *assertions* that are in the form of semialgebraic set memberships. Using graph models, this is done as follows:

$$\text{at location } i : \text{ assert } x \in A_i \Rightarrow \text{ define } X_- = \{(i, x) \in X \mid x \in X \setminus A_i\},$$

$$\text{at location } i : \text{ assert } x \notin A_i \Rightarrow \text{ define } X_- = \{(i, x) \in X \mid x \in A_i\}.$$

In particular, safety assertions for division-by-zero or taking the square root (or logarithm) of positive variables are standard and must be automatically included in numerical programs (cf. Sec. III-A, Table I).

4) *Program Invariants:* A program invariant is a property that holds throughout the execution of the program. The property indicates that the variables reside in a semialgebraic subset $X_I \subset X$. Essentially, any method that is used for verifying unreachability of a subset $X_- \subset X$, can be applied for verifying invariance of X_I by defining $X_- = X \setminus X_I$, and vice versa.

D. Implications of the Abstractions

For mathematical correctness, we must show that if an \mathcal{A} -representation of a program satisfies the unreachability and FTT specifications, then so does the \mathcal{C} -representation, i.e., the actual program. This is established in the following proposition.

Proposition 2: Let $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ be an \mathcal{A} -representation of program \mathcal{P} with \mathcal{C} -representation $\mathcal{S}(X, f, X_0, X_\infty)$. Let $X_- \subset X$ and $\overline{X}_- \subset \overline{X}$ be such that $X_- \subseteq \overline{X}_-$. Assume that the unreachability property w.r.t. \overline{X}_- has been verified for $\overline{\mathcal{S}}$. Then, \mathcal{P} satisfies the unreachability property w.r.t. X_- . Moreover, if the FTT property holds for $\overline{\mathcal{S}}$, then \mathcal{P} terminates in finite time.

Proof: See Appendix II. ■

Since we are not concerned with undecidability issues, and in light of Proposition 2, we will not differentiate between abstract or concrete representations in the remainder of this paper.

III. LYAPUNOV INVARIANTS AS BEHAVIOR CERTIFICATES

Analogous to a Lyapunov function, a Lyapunov invariant is a real-valued function of the program variables satisfying a *difference inequality* along the execution trace.

Definition 4: A (θ, μ) -Lyapunov invariant for $\mathcal{S}(X, f, X_0, X_\infty)$ is a function $V : X \mapsto \mathbb{R}$ such that

$$V(x_+) - \theta V(x) \leq -\mu \quad \forall x \in X, x_+ \in f(x) : x \notin X_\infty. \quad (9)$$

where $(\theta, \mu) \in [0, \infty) \times [0, \infty)$. Thus, a Lyapunov invariant satisfies the *difference inequality* (9) along the trajectories of \mathcal{S} until they reach a terminal state X_∞ .

It follows from Definition 4 that a Lyapunov invariant is not necessarily nonnegative, or bounded from below, and in general it need not be monotonically decreasing. While the zero level set of V defines an invariant set in the sense that $V(x_k) \leq 0$ implies $V(x_{k+l}) \leq 0$, for all $l \geq 0$, monotonicity depends on θ and the initial condition. For instance, if $V(x_0) \leq 0, \forall x_0 \in X_0$, then (9) implies that $V(x) \leq 0$ along the trajectories of \mathcal{S} , however, $V(x)$ may not be monotonic if $\theta < 1$, though it will be monotonic for $\theta \geq 1$. Furthermore, the level sets of a Lyapunov invariant need not be bounded closed curves.

Proposition 3 (to follow) formalizes the interpretation of Definition 4 for the specific modeling languages. Natural Lyapunov invariants for graph models are functions of the form

$$V(\tilde{x}) \equiv V(i, x) = \sigma_i(x), \quad i \in N, \quad (10)$$

which assign a polynomial Lyapunov function to every node $i \in \mathcal{N}$ on the graph $G(\mathcal{N}, \mathcal{E})$.

Proposition 3: Let $\mathcal{S}(F, H, X_0, n, n_w, n_v)$ and properly labeled graph $G(\mathcal{N}, \mathcal{E})$ be the MIL and graph models for a computer program \mathcal{P} . The function $V : [-1, 1]^n \mapsto \mathbb{R}$ is a (θ, μ) -Lyapunov invariant for \mathcal{P} if it satisfies:

$$V(Fx_e) - \theta V(x) \leq -\mu, \quad \forall (x, x_e) \in [-1, 1]^n \times \Xi,$$

where

$$\Xi = \{(x, w, v, 1) \mid H \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, \quad (w, v) \in [-1, 1]^{n_w} \times \{-1, 1\}^{n_v}\}.$$

The function $V : \mathcal{N} \times \mathbb{R}^n \mapsto \mathbb{R}$, satisfying (10) is a (θ, μ) -Lyapunov invariant for \mathcal{P} if

$$\sigma_j(x_+) - \theta \sigma_i(x) \leq -\mu, \quad \forall (i, j, k) \in \mathcal{E}, \quad (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times \overline{T}_{ji}^k x. \quad (11)$$

Note that a generalization of (9) allows for θ and μ to depend on the state x , although simultaneous search for $\theta(x)$ and $V(x)$ leads to non-convex conditions, unless the dependence of θ on x is fixed a-priori. We allow for dependence of θ on the discrete component of the state in the following way:

$$\sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) \leq -\mu_{ji}, \quad \forall (i, j, k) \in \mathcal{E}, \quad (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times \overline{T}_{ji}^k x \quad (12)$$

A. Behavior Certificates

1) Finite-Time Termination (FTT) Certificates: The following proposition is applicable to FTT analysis of both finite and infinite state models.

Proposition 4: Finite-Time Termination. Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. If there exists a (θ, μ) -Lyapunov invariant $V : X \mapsto \mathbb{R}$, uniformly bounded on $X \setminus X_\infty$, satisfying (9) and the following conditions

$$V(x) \leq -\eta \leq 0, \quad \forall x \in X_0 \quad (13)$$

$$\mu + (\theta - 1) \|V\|_\infty > 0 \quad (14)$$

$$\max(\mu, \eta) > 0 \quad (15)$$

where $\|V\|_\infty = \sup_{x \in X \setminus X_\infty} V(x) < \infty$, then \mathcal{P} terminates in finite time, and an upper-bound on the number

of iterations is given by

$$T_u = \begin{cases} \frac{\log(\mu + (\theta - 1) \|V\|_\infty) - \log(\mu)}{\log \theta} & , \quad \theta \neq 1, \mu > 0 \\ \frac{\log(\|V\|_\infty) - \log(\eta)}{\log \theta} & , \quad \theta \neq 1, \mu = 0 \\ \|V\|_\infty / \mu & , \quad \theta = 1 \end{cases} \quad (16)$$

Proof: The proof is presented in Appendix II. ■

When the state-space X is finite, or when the Lyapunov invariant V is only a function of a subset of the variables that assume values in a finite set, e.g., integer counters, it follows from Proposition 4 that V being a (θ, μ) -Lyapunov invariant for any $\theta \geq 1$ and $\mu > 0$ is sufficient for certifying FTT, and uniform boundedness of V need not be established a-priori.

Example 3: Consider the IntegerDivision program presented in Example 1. The function $V : X \mapsto \mathbb{R}$, defined according to $V : (\text{dd}, \text{dr}, \text{q}, \text{r}) \mapsto \text{r}$ is a $(1, \text{dr})$ -Lyapunov invariant for IntegerDivision: at every step, V decreases by $\text{dr} > 0$. Since X is finite, the program IntegerDivision terminates in finite time. This, however, only proves absence of infinite loops. The program could terminate with an overflow.

2) *Separating Manifolds and Certificates of Boundedness:* Let V be a Lyapunov invariant satisfying (9) with $\theta = 1$. The level sets of V , defined by $\mathcal{L}_r(V) \stackrel{\text{def}}{=} \{x \in X : V(x) < r\}$, are invariant with respect to (1) in the sense that $x(t+1) \in \mathcal{L}_r(V)$ whenever $x(t) \in \mathcal{L}_r(V)$. However, for $r = 0$, the level sets $\mathcal{L}_r(V)$ remain invariant with respect to (1) for any nonnegative θ . This is an important property with the implication that $\theta = 1$ (i.e., monotonicity) is not necessary for establishing a separating manifold between the reachable set and the unsafe regions of the state space (cf. Theorem 1).

Theorem 1: Lyapunov Invariants as Separating Manifolds. Let \mathcal{V} denote the set of all (θ, μ) -Lyapunov invariants satisfying (9) for program $\mathcal{P} \equiv \mathcal{S}(X, f, X_0, X_\infty)$. Let I be the identity map, and for $h \in \{f, I\}$ define

$$h^{-1}(X_-) = \{x \in X \mid h(x) \cap X_- \neq \emptyset\}.$$

A subset $X_- \subset X$, where $X_- \cap X_0 = \emptyset$ can never be reached along the trajectories of \mathcal{P} , if there exists

$V \in \mathcal{V}$ satisfying

$$\sup_{x \in X_0} V(x) < \inf_{x \in h^{-1}(X_-)} V(x) \quad (17)$$

and either $\theta = 1$, or one of the following two conditions hold:

$$(I) \quad \theta < 1 \quad \text{and} \quad \inf_{x \in h^{-1}(X_-)} V(x) > 0. \quad (18)$$

$$(II) \quad \theta > 1 \quad \text{and} \quad \sup_{x \in X_0} V(x) \leq 0. \quad (19)$$

Proof: The proof is presented in Appendix II. ■

The following corollary is based on Theorem 1 and Proposition 4 and presents computationally implementable criteria (cf. Section IV) for simultaneously establishing FTT and absence of overflow.

Corollary 1: Overflow and FTT Analysis Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Let $\alpha > 0$ be a diagonal matrix specifying the overflow limit, and let $X_- = \{x \in X \mid \|\alpha^{-1}x\|_\infty > 1\}$. Let $q \in \mathbb{N} \cup \{\infty\}$, $h \in \{f, I\}$, and let the function $V : X \mapsto \mathbb{R}$ be a (θ, μ) -Lyapunov invariant for \mathcal{S} satisfying

$$V(x) \leq 0 \quad \forall x \in X_0. \quad (20)$$

$$V(x) \geq \sup \left\{ \|\alpha^{-1}h(x)\|_q - 1 \right\} \quad \forall x \in X. \quad (21)$$

Then, an *overflow runtime error* will not occur during any execution of \mathcal{P} . In addition, if $\mu > 0$ and $\mu + \theta > 1$, then, \mathcal{P} terminates in at most T_u iterations where $T_u = \mu^{-1}$ if $\theta = 1$, and for $\theta \neq 1$ we have:

$$T_u = \frac{\log(\mu + (\theta - 1)\|V\|_\infty) - \log \mu}{\log \theta} \leq \frac{\log(\mu + \theta - 1) - \log \mu}{\log \theta} \quad (22)$$

where $\|V\|_\infty = \sup_{x \in X \setminus \{X_- \cup X_\infty\}} |V(x)|$.

Proof: The proof is presented in Appendix II. ■

Remarks

- Application of Corollary 1 with $h = f$ typically leads to much less conservative results compared with $h = I$, though the computational costs are also higher.

- An alternative, less conservative formulation is obtained when (21) is replaced by n constraints in the following way:

$$V(x) \geq \sup \left\{ \left| \alpha_k^{-1} h(x)_k \right| - 1 \right\} \quad \forall x \in X, \quad k \in \mathbb{Z}(1, n) \quad (23)$$

where α_k is the overflow limit for scalar variable x_k . Since $\alpha_k^{-1} x_k$ is a scalar quantity, $\left\| \alpha_k^{-1} h(x)_k \right\|_q = \left| \alpha_k^{-1} h(x)_k \right|$ and (23) can be used in conjunction with quadratic Lyapunov invariants and the \mathcal{S} -Procedure for convex relaxation to formulate the problem of searching for V as semidefinite optimization problem (cf. Section IV). However, (23) is computationally more expensive than (21), as it increases the number of constraints.

- An even less conservative but computationally more demanding alternative would be to construct a different Lyapunov invariant for each variable and rewrite (23), (21), (20) along with (9), for n different functions $V_k(\cdot)$. For instance:

$$V_k(x) \leq 0, \quad \forall x \in X_0.$$

$$V_k(x) \geq \left| \alpha_k^{-1} h(x)_k \right| - 1 \quad \forall x_k \in X.$$

a) General Unreachability and FTT Analysis over Graph Models: The results presented so far in this section (Theorem 1, Corollary 1, and Proposition 4) are readily applicable to MILMs. These results will be applied in Section IV to formulate the verification problem as a convex optimization problem. Herein, we present an adaptation of these results to analysis of graph models.

Definition 5: A cycle \mathcal{C}_m on a graph $G(\mathcal{N}, \mathcal{E})$ is an ordered list of m triplets $(n_1, n_2, k_1), (n_2, n_3, k_2), \dots, (n_m, n_{m+1}, k_m)$, where $n_{m+1} = n_1$, and $(n_j, n_{j+1}, k_j) \in \mathcal{E}, \forall j \in \mathbb{Z}(1, m)$. A simple cycle is a cycle with no strict sub-cycles.

Corollary 2: Unreachability and FTT Analysis of Graph Models. Consider a program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V(i, x) = \sigma_i(x)$ be a Lyapunov invariant for $G(\mathcal{N}, \mathcal{E})$, satisfying (12) and

$$\sigma_\emptyset(x) \leq 0, \quad \forall x \in X_\emptyset \quad (24)$$

and either one of the following two conditions:

$$(I) : \sigma_i(x) > 0, \quad \forall x \in X_i \cap X_{i-}, \quad i \in \mathcal{N} \setminus \{\emptyset\} \quad (25)$$

$$(II) : \sigma_i(x) > 0, \quad \forall x \in X_j \cap T^{-1}(X_{i-}), \quad i \in \mathcal{N} \setminus \{\emptyset\}, \quad j \in \mathcal{I}(i), \quad T \in \{\bar{T}_{ij}^k\} \quad (26)$$

where

$$T^{-1}(X_{i-}) = \{x \in X_i | T(x) \cap X_{i-} \neq \emptyset\}$$

Then, \mathcal{P} satisfies the unreachability property w.r.t. the collection of sets X_{i-} , $i \in \mathcal{N} \setminus \{\emptyset\}$. In addition, if for every simple cycle $\mathcal{C} \in G$, we have:

$$(\theta(\mathcal{C}) - 1) \|\sigma(\mathcal{C})\|_\infty + \mu(\mathcal{C}) > 0, \text{ and } \mu(\mathcal{C}) > 0, \text{ and } \|\sigma(\mathcal{C})\|_\infty < \infty, \quad (27)$$

where

$$\theta(\mathcal{C}) = \prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k, \quad \mu(\mathcal{C}) = \max_{(i,j,k) \in \mathcal{C}} \mu_{ij}^k, \quad \|\sigma(\mathcal{C})\|_\infty = \max_{(i,.,.) \in \mathcal{C}} \sup_{x \in X_i \setminus X_{i-}} |\sigma_i(x)| \quad (28)$$

then \mathcal{P} terminates in at most T_u iterations where

$$T_u = \sum_{\mathcal{C} \in G: \theta(\mathcal{C}) \neq 1} \frac{\log((\theta(\mathcal{C}) - 1) \|\sigma(\mathcal{C})\|_\infty + \mu(\mathcal{C})) - \log \mu(\mathcal{C})}{\log \theta(\mathcal{C})} + \sum_{\mathcal{C} \in G: \theta(\mathcal{C}) = 1} \frac{\|\sigma(\mathcal{C})\|_\infty}{\mu(\mathcal{C})}.$$

Proof: The proof is presented in Appendix II. ■

For verification against an overflow violation specified by a diagonal matrix $\alpha > 0$, Corollary 2 is applied with $X_- = \{x \in \mathbb{R}^n \mid \|\alpha^{-1}x\|_\infty > 1\}$. Hence, (25) becomes $\sigma_i(x) \geq p(x)(\|\alpha^{-1}x\|_q - 1)$, $\forall x \in X_i$, $i \in \mathcal{N} \setminus \{\emptyset\}$, where $p(x) > 0$. User-specified assertions, as well as many other standard safety specifications, such as absence of division-by-zero can be verified using Corollary 2 (See Table I).

– *Identification of Dead Code:* Suppose that we wish to verify that a discrete location $i \in \mathcal{N} \setminus \{\emptyset\}$ in a graph model $G(\mathcal{N}, \mathcal{E})$ is unreachable. If a function satisfying the criteria of Corollary 2 with $X_{i-} = \mathbb{R}^n$ can be found, then location i can never be reached. Condition (25) then becomes $\sigma_i(x) \geq 0$, $\forall x \in \mathbb{R}^n$.

TABLE I
APPLICATION OF COROLLARY 2 TO THE VERIFICATION OF VARIOUS SAFETY SPECIFICATIONS.

			apply Corollary 2 with:
At location i :	assert $x \in X_a$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x \in \mathbb{R}^n \setminus X_a\}$
At location i :	assert $x \notin X_a$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x \in X_a\}$
At location i :	(expr.)/ x_o	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o = 0\}$
At location i :	$\sqrt[k]{x_o}$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o < 0\}$
At location i :	$\log(x_o)$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o \leq 0\}$
At location i :	dead code	\Rightarrow	$X_{i-} := \mathbb{R}^n$

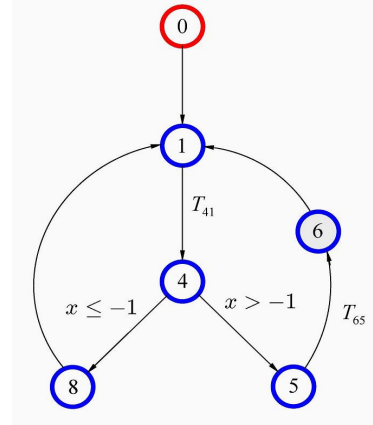
Example 4: Consider the following program

```

void ComputeTurnRate (void)
L0 : {double x = {0}; double y = {*PtrToY};
L1 : while (1)
L2 : {   y = *PtrToY;
L3 :     x = (5 * sin(y) + 1)/3;
L4 :     if x > -1 {
L5 :       x = x + 1.0472;
L6 :       TurnRate = y/x; }
L7 :     else {
L8 :       TurnRate = 100 * y/3.1416 }}

```

Program 3



Graph of an abstraction of Program 3

Note that x can be zero right after the assignment $x = (5 \sin(y) + 1)/3$. However, at location L6, x cannot be zero and division-by-zero will not occur. The graph model of an abstraction of Program 3 is shown next to the program and is defined by the following elements: $T_{65} : x \mapsto x + 1.0472$, and $T_{41} : x \mapsto [-4/3, 2]$. The other transition labels are identity. The only non-universal passport labels are Π_{54} and Π_{84} as shown in the figure. Define

$$\sigma_6(x) = -x^2 - 100x + 1, \quad \sigma_5(x) = -(x + 1309/1250)^2 - 100x - 2543/25$$

$$\sigma_0(x) = \sigma_1(x) = \sigma_4(x) = \sigma_8(x) = -x^2 + 2x - 3.$$

It can be verified that $V(x) = \sigma_i(x)$ is a $(\theta, 1)$ -Lyapunov invariant for Program 3 with variable rates:

$\theta_{65} = 1$, and $\theta_{ij} = 0$ for all $(i, j) \neq (6, 5)$. Since

$$-2 = \sup_{x \in X_0} \sigma_0(x) < \inf_{x \in X_-} \sigma_6(x) = 1$$

the state $(6, x = 0)$ cannot be reached. Hence, a division by zero will never occur. We will show in the next section how to find such functions in general.

IV. COMPUTATION OF LYAPUNOV INVARIANTS

It is well known that the main difficulty in using Lyapunov functions in system analysis is finding them. Naturally, using Lyapunov invariants in software analysis inherits the same difficulties. However, the recent advances in hardware and software technology, e.g., semi-definite programming [22], [61], and linear programming software [23] present an opportunity for new approaches to software verification based on numerical optimization.

A. Preliminaries

1) *Convex Parameterization of Lyapunov Invariants:* The chances of finding a Lyapunov invariant are increased when (9) is only required on a subset of $X \setminus X_\infty$. For instance, for $\theta \leq 1$, it is tempting to replace (9) with

$$V(x_+) - \theta V(x) \leq -\mu, \quad \forall x \in X \setminus X_\infty : V(x) < 1, \quad x_+ \in f(x) \quad (29)$$

In this formulation V is not required to satisfy (9) for those states which cannot be reached from X_0 . However, the set of all functions $V : X \mapsto \mathbb{R}$ satisfying (29) is not convex and finding a solution for (29) is typically much harder than (9). Such non-convex formulations are not considered in this paper.

The first step in the search for a function $V : X \mapsto \mathbb{R}$ satisfying (9) is selecting a finite-dimensional linear parameterization of a candidate function V :

$$V(x) = V_\tau(x) = \sum_{k=1}^n \tau_k V_k(x), \quad \tau = (\tau_k)_{k=1}^n, \quad \tau_k \in \mathbb{R}, \quad (30)$$

where $V_k : X \mapsto \mathbb{R}$ are fixed basis functions. Next, for every $\tau = (\tau_k)_{k=1}^n$ let

$$\phi(\tau) = \max_{x \in X \setminus X_\infty, \quad x_+ \in f(x)} V_\tau(x_+) - \theta V_\tau(x),$$

(assuming for simplicity that the maximum does exist). Since $\phi(\cdot)$ is a maximum over x of a family of

linear functions in τ , $\phi(\cdot)$ is a convex function. If minimizing $\phi(\cdot)$ over the unit disk yields a negative minimum, the optimal τ^* defines a valid Lyapunov invariant $V_{\tau^*}(x)$. Otherwise, no linear combination (30) yields a valid solution for (9).

The success and efficiency of the proposed approach depend on computability of $\phi(\cdot)$ and its subgradients. While $\phi(\cdot)$ is convex, the same does not necessarily hold for $V_{\tau}(x_+) - \theta V_{\tau}(x)$ as a function of x . In fact, if $X \setminus X_{\infty}$ is non-convex, which is often the case even for very simple programs, computation of $\phi(\cdot)$ becomes a non-convex optimization problem even if $V_{\tau}(x_+) - V_{\tau}(x)$ is a nice (e.g. linear or concave and smooth) function of x . To get around this hurdle, we propose using convex relaxation techniques which essentially lead to computation of a convex upper bound for $\phi(\tau)$.

2) *Convex Relaxation Techniques*: Such techniques constitute a broad class of techniques for constructing finite-dimensional, convex approximations for difficult non-convex optimization problems. Some of the results most relevant to the software verification framework presented in this paper can be found in [36] for SDP relaxation of binary integer programs, [39] and [47] for SDP relaxation of quadratic programs, [64] for \mathcal{S} -Procedure in robustness analysis, and [50],[49] for sum-of-squares relaxation in polynomial non-negativity verification. We provide a brief overview of the latter two techniques.

a) *The \mathcal{S} -Procedure* : The \mathcal{S} -Procedure is commonly used for construction of Lyapunov functions for nonlinear dynamical systems. Let functions $\phi_i : X \mapsto \mathbb{R}$, $i \in \mathbb{Z}(0, m)$, and $\psi_j : X \mapsto \mathbb{R}$, $j \in \mathbb{Z}(1, n)$ be given, and suppose that we are concerned with evaluating the following assertions:

$$(I): \phi_0(x) > 0, \forall x \in \{x \in X \mid \phi_i(x) \geq 0, \psi_j(x) = 0, i \in \mathbb{Z}(1, m), j \in \mathbb{Z}(1, n)\} \quad (31)$$

$$(II): \exists \tau_i \in \mathbb{R}^+, \exists \mu_j \in \mathbb{R}, \text{ such that } \phi_0(x) > \sum_{i=1}^m \tau_i \phi_i(x) + \sum_{j=1}^n \mu_j \psi_j(x). \quad (32)$$

The implication $(II) \rightarrow (I)$ is trivial. The process of replacing assertion (I) by its *relaxed* version (II) is called the \mathcal{S} -Procedure. Note that condition (II) is convex in decision variables τ_i and μ_j . The implication $(I) \rightarrow (II)$ is generally not true and the \mathcal{S} -Procedure is called lossless for special cases where (I) and (II) are equivalent. A well-known such case is when $m = 1$, $n = 0$, and ϕ_0, ϕ_1 are quadratic functionals. A comprehensive discussion of the \mathcal{S} -Procedure as well as available results on its losslessness can be found

in [26]. Other variations of \mathcal{S} -Procedure with non-strict inequalities exist as well.

b) Sum-of-Squares (SOS) Relaxation : The SOS relaxation technique can be interpreted as the generalized version of the \mathcal{S} -Procedure and is concerned with verification of the following assertion:

$$f_j(x) \geq 0, \forall j \in J, \quad g_k(x) \neq 0, \forall k \in K, \quad h_l(x) = 0, \forall l \in L \Rightarrow -f_0(x) \geq 0, \quad (33)$$

where f_j, g_k, h_l are polynomial functions. It is easy to see that the problem is equivalent to verification of emptiness of a semialgebraic set, a necessary and sufficient condition for which is given by the Positivstellensatz Theorem [9]. In practice, sufficient conditions in the form of nonnegativity of polynomials are formulated. The non-negativity conditions are in turn relaxed to SOS conditions. Let $\Sigma[y_1, \dots, y_m]$ denote the set of SOS polynomials in m variables y_1, \dots, y_m , i.e. the set of polynomials that can be represented as $p = \sum_{i=1}^t p_i^2$, $p_i \in \mathbb{P}_m$, where \mathbb{P}_m is the polynomial ring of m variables with real coefficients. Then, a sufficient condition for (33) is that there exist SOS polynomials $\tau_0, \tau_i, \tau_{ij} \in \Sigma[x]$ and polynomials ρ_l , such that

$$\tau_0 + \sum_i \tau_i f_i + \sum_{i,j} \tau_{ij} f_i f_j + \sum_l \rho_l h_l + \left(\prod g_k\right)^2 = 0$$

Matlab toolboxes SOSTOOLS [54], or YALMIP [35] automate the process of converting an SOS problem to an SDP, which is subsequently solved by available software packages such as LMILAB [22], or SeDumi [61]. Interested readers are referred to [49], [40], [50], [54] for more details.

B. Optimization of Lyapunov Invariants for Mixed-Integer Linear Models

Natural Lyapunov invariant candidates for MILMs are quadratic and affine functionals.

1) Quadratic Invariants: The linear parameterization of the space of quadratic functionals mapping \mathbb{R}^n to \mathbb{R} is given by:

$$\mathcal{V}_x^2 = \left\{ V : \mathbb{R}^n \mapsto \mathbb{R} \mid V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix}, P \in \mathbb{S}^{n+1} \right\}, \quad (34)$$

where \mathbb{S}^n is the set of n -by- n symmetric matrices. We have the following lemma.

Lemma 1: Consider a program \mathcal{P} and its MILM $\mathcal{S}(F, H, X_0, n, n_w, n_v)$. The program admits a quadratic (θ, μ) -Lyapunov invariant $V \in \mathcal{V}_x^2$, if there exists a matrix $Y \in \mathbb{R}^{n_e \times n_H}$, $n_e = n + n_w + n_v + 1$, a diagonal matrix $D_v \in \mathbb{D}^{n_v}$, a positive semidefinite diagonal matrix $D_{xw} \in \mathbb{D}_+^{n+n_w}$, and a symmetric

matrix $P \in \mathbb{S}^{n+1}$, satisfying the following LMIs:

$$L_1^T P L_1 - \theta L_2^T P L_2 \preceq \text{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - (\lambda + \mu) L_5^T L_5$$

$$\lambda = \text{Trace } D_{xw} + \text{Trace } D_v$$

where

$$L_1 = \begin{bmatrix} F \\ L_5 \end{bmatrix}, \quad L_2 = \begin{bmatrix} I_n & 0_{n \times (n_e - n)} \\ 0_{1 \times (n_e - 1)} & 1 \end{bmatrix}, \quad L_3 = \begin{bmatrix} I_{n+n_w} \\ 0_{(n_v+1) \times (n+n_w)} \end{bmatrix}^T, \quad L_4 = \begin{bmatrix} 0_{(n+n_w) \times n_v} \\ I_{n_v} \\ 0_{1 \times n_v} \end{bmatrix}^T, \quad L_5 = \begin{bmatrix} 0_{(n_e-1) \times 1} \\ 1 \end{bmatrix}^T$$

Proof: The proof is presented in Appendix II ■

The following theorem summarizes our results for verification of absence of overflow and/or FTT for MILMs. The result follows from Lemma 1 and Corollary 1 with $q = 2$, $h = f$, though the theorem is presented without a detailed proof.

Theorem 2: Optimization-Based MILM Verification. Let $\alpha : 0 \prec \alpha \preceq I_n$ be a diagonal positive definite matrix specifying the overflow limit. An overflow runtime error does not occur during any execution of \mathcal{P} if there exist matrices $Y_i \in \mathbb{R}^{n_e \times n_H}$, diagonal matrices $D_{iv} \in \mathbb{D}^{n_v}$, positive semidefinite diagonal matrices $D_{ixw} \in \mathbb{D}_+^{n+n_w}$, and a symmetric matrix $P \in \mathbb{S}^{n+1}$ satisfying the following LMIs:

$$\begin{bmatrix} x_0 & 1 \end{bmatrix} P \begin{bmatrix} x_0 & 1 \end{bmatrix}^T \leq 0, \quad \forall x_0 \in X_0 \quad (35)$$

$$L_1^T P L_1 - \theta L_2^T P L_2 \preceq \text{He}(Y_1 H) + L_3^T D_{1xw} L_3 + L_4^T D_{1v} L_4 - (\lambda_1 + \mu) L_5^T L_5 \quad (36)$$

$$L_1^T \Lambda L_1 - L_2^T P L_2 \preceq \text{He}(Y_2 H) + L_3^T D_{2xw} L_3 + L_4^T D_{2v} L_4 - \lambda_2 L_5^T L_5 \quad (37)$$

where $\Lambda = \text{diag}\{\alpha^{-2}, -1\}$, $\lambda_i = \text{Trace } D_{ixw} + \text{Trace } D_{iv}$, and $0 \preceq D_{ixw}$, $i = 1, 2$. In addition, if $\mu + \theta > 1$, then \mathcal{P} terminates in a most T_u steps where T_u is given in (22).

2) Affine Invariants: Affine Lyapunov invariants can often establish strong properties, e.g., boundedness, for variables with simple uncoupled dynamics (e.g. counters) at a low computational cost. For variables with more complicated dynamics, affine invariants may simply establish sign-invariance (e.g., $x_i \geq 0$) or more generally, upper or lower bounds on some linear combination of certain variables. As we will observe in Section V, establishing these simple behavioral properties is important as they can be recursively added to the model (e.g., the matrix H in a MILM, or the invariant sets X_i in a graph

model) to improve the chances of success in proving stronger properties via higher order invariants. The linear parameterization of the subspace of linear functionals mapping \mathbb{R}^n to \mathbb{R} , is given by:

$$\mathcal{V}_x^1 = \left\{ V : \mathbb{R}^n \mapsto \mathbb{R} \mid V(x) = K^T [x \ 1]^T, K \in \mathbb{R}^{n+1} \right\}. \quad (38)$$

It is possible to search for the affine invariants via semidefinite programming or linear programming.

Proposition 5: SDP Characterization of Linear Invariants: There exists a (θ, μ) -Lyapunov invariant $V \in \mathcal{V}_x^1$ for a program $\mathcal{P} \equiv \mathcal{S}(F, H, X_0, n, n_w, n_v)$, if there exists a matrix $Y \in \mathbb{R}^{n_e \times n_H}$, a diagonal matrix $D_v \in \mathbb{D}^{n_v}$, a positive semidefinite diagonal matrix $D_{xw} \in \mathbb{D}_+^{(n+n_w) \times (n+n_w)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying the following LMI:

$$\text{He}(L_1^T K L_5 - \theta L_5^T K^T L_2) \prec \text{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - (\lambda + \mu) L_5^T L_5 \quad (39)$$

where $\lambda = \text{Trace } D_{xw} + \text{Trace } D_v$ and $0 \preceq D_{xw}$.

Proposition 6: LP Characterization of Linear Invariants: There exists a (θ, μ) -Lyapunov invariant for a program $\mathcal{P} \equiv \mathcal{S}(F, H, X_0, n, n_w, n_v)$ in the class \mathcal{V}_x^1 , if there exists a matrix $Y \in \mathbb{R}^{1 \times n_H}$, and nonnegative matrices $\underline{D}_v, \overline{D}_v \in \mathbb{R}^{1 \times n_v}$, $\underline{D}_{xw}, \overline{D}_{xw} \in \mathbb{R}^{1 \times (n+n_w)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying:

$$K^T L_1 - \theta K^T L_2 - YH - (\underline{D}_{xw} - \overline{D}_{xw}) L_3 - (\underline{D}_v - \overline{D}_v) L_4 - (D_1 + \mu) L_5 = 0 \quad (40a)$$

$$D_1 + (\overline{D}_v + \underline{D}_v) \mathbf{1}_r + (\overline{D}_{xw} + \underline{D}_{xw}) \mathbf{1}_{n+n_w} \leq 0 \quad (40b)$$

$$\overline{D}_v, \underline{D}_v, \overline{D}_{xw}, \underline{D}_{xw} \geq 0 \quad (40c)$$

where D_1 is either 0 or -1 . As a special case of (40), a subset of all the affine invariants is characterized by the set of all solutions of the following system of linear equations:

$$K^T L_1 - \theta K^T L_2 + L_5 = 0 \quad (41)$$

Remark 1: When the objective is to establish properties of the form $Kx \geq a$ for a fixed K , (e.g., when establishing sign-invariance for certain variables), matrix K in (39)–(41) is fixed and thus one can make θ a decision variable subject to $\theta \geq 0$. Exploiting this convexity is extremely helpful for successfully establishing such properties.

The advantage of using SDP for computation of linear invariants is that efficient SDP relaxations for treatment of binary variables exists [39], [47], [25], though the computational costs are typically higher than the LP-based approaches. In contrast, linear programming relaxations of the binary constraints are more involved than the corresponding SDP relaxations. Two extreme remedies can be readily considered. The first is to relax the binary constraints and treat the variables as continuous variables. The second is to consider each of the 2^{n_v} different possibilities (one for each vertex of $\{-1, 1\}^{n_v}$) separately. This approach is practical only if n_v is small. More sophisticated schemes can be developed based on hierarchical relaxations or convex hull approximations of binary integer programs [60], [36].

C. Optimization of Lyapunov Invariants for Graph Models

A linear parameterization of the subspace of polynomial functionals with total degree less than or equal to d is given by:

$$\mathcal{V}_x^d = \left\{ V : \mathbb{R}^n \mapsto \mathbb{R} \mid V(x) = K^T Z(x), K \in \mathbb{R}^N, N = \binom{n+d}{d} \right\} \quad (42)$$

where $Z(x)$ is a vector of length $\binom{n+d}{d}$, consisting of all monomials of degree less than or equal to d in n variables x_1, \dots, x_n . A linear parametrization of Lyapunov invariants for graph models is defined according to (10), where for every $i \in \mathcal{N}$, we have $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$, where $d(i)$ is a selected degree bound for $\sigma_i(\cdot)$. Depending on the dynamics of the model, the degree bounds $d(i)$, and the convex relaxation technique, the corresponding optimization problem will become a linear, semidefinite, or SOS optimization problem.

1) Node-wise Polynomial Invariants: We present generic conditions for verification over graph models using SOS programming. For the specific case of verification of *linear graph models* via quadratic invariants and the \mathcal{S} -Procedure for relaxation of non-convex constraints we present explicit LMIs. The corresponding Matlab code is available at [65]. The following theorem follows from Corollary 2.

Theorem 3: Optimization-Based Graph Model Verification. Consider a program \mathcal{P} , and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V : \Omega^n \mapsto \mathbb{R}$, be given by (10), where $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$. Then, the functions $\sigma_i(\cdot)$, $i \in \mathcal{N}$ define a Lyapunov invariant for \mathcal{P} , if for all $(i, j, k) \in \mathcal{E}$ we have:

$$-\sigma_j(T_{ji}^k(x, w)) + \theta_{ji}^k \sigma_i(x) - \mu_{ji}^k \in \Sigma[x, w] \text{ subject to } (x, w) \in ((X_i \cap \Pi_{ji}^k) \times [-1, 1]^{n_w}) \cap S_{ji}^k \quad (43)$$

Furthermore, \mathcal{P} satisfies the unreachability property w.r.t. the collection of sets X_{i-} , $i \in \mathcal{N} \setminus \{\emptyset\}$, if there exist $\varepsilon_i \in (0, \infty)$, $i \in \mathcal{N} \setminus \{\emptyset\}$, such that

$$-\sigma_\emptyset(x) \in \Sigma[x] \text{ subject to } x \in X_\emptyset \quad (44)$$

$$\sigma_i(x) - \varepsilon_i \in \Sigma[x] \text{ subject to } x \in X_i \cap X_{i-}, \quad i \in \mathcal{N} \setminus \{\emptyset\} \quad (45)$$

As discussed in Section IV-A2b, the SOS relaxation techniques can be applied for formulating the search problem for functions σ_i satisfying (43)–(45) as a convex optimization problem. For instance, if

$$((X_i \cap \Pi_{ji}^k) \times [-1, 1]^{n_w}) \cap S_{ji}^k = \{(x, w) \mid f_p(x, w) \geq 0, \quad h_l(x, w) = 0\},$$

then, (43) can be relaxed as an SOS optimization problem of the following form:

$$-\sigma_j(T_{ji}^k(x, w)) + \theta_{ji}^k \sigma_i(x) - \mu_{ji}^k - \sum_p \tau_p f_p - \sum_{p,q} \tau_{pq} f_p f_q - \sum_l \rho_l h_l \in \Sigma[x, w], \quad \text{s.t. } \tau_p, \tau_{pq} \in \Sigma[x, w].$$

Software packages such as SOSTOOLS [54] or YALMIP [35] can then be used for formulating the SOS optimization problems as semidefinite programs.

2) Node-wise Quadratic Invariants for Quasi-Linear Graph Models: In a quasi-linear graph model, all the transition labels are linear. The passport labels and the invariant sets are defined by linear and/or quadratic functions. Let $\bar{x} = [x \quad 1]^T$, $\bar{n} = n + 1$, and let, for all $(i, j, k) \in \mathcal{E}$, a compact description of the set $X_i \cap \Pi_{ji}^k$ be given by:

$$X_i \cap \Pi_{ji}^k = \{x \in \mathbb{R}^n \mid \bar{x}^T Q_{ji,l}^k \bar{x} \geq 0, \quad \bar{x}^T R_{ji,m}^k \bar{x} = 0, \quad G_{ji}^k \bar{x} \geq 0, \quad H_{ji}^k \bar{x} = 0\}, \quad (46)$$

where $G_{ji}^k \in \mathbb{R}^{g_{ji}^k \times \bar{n}}$, $H_{ji}^k \in \mathbb{R}^{h_{ji}^k \times \bar{n}}$, $Q_{ji,s}^k \in \mathbb{R}^{\bar{n} \times \bar{n}}$, $s \in S_{ji}^k$, and $R_{ji,m}^k \in \mathbb{R}^{\bar{n} \times \bar{n}}$, $m \in M_{ji}^k$, where S_{ji}^k and M_{ji}^k are some sets. For all $(i, j, k) \in \mathcal{E}$ let the transitions be given by $T_{ji}^k x = A_{ji}^k x + B_{ji}^k w + E_{ji}^k$, where $w \in [-1, 1]^{q_{ji}^k}$. Before proceeding to the next theorem, for convenience, we define the following matrices:

$$F_{ji}^k = \begin{bmatrix} A_{ji}^k & B_{ji}^k & E_{ji}^k \\ 0_{1 \times n} & 0_{1 \times q_{ji}^k} & 1 \end{bmatrix}, \quad L_{ji}^k = \begin{bmatrix} I_n & 0_{n \times q_{ji}^k} & 0_{n \times 1} \\ 0_{1 \times n} & 0_{1 \times q_{ji}^k} & 1 \end{bmatrix} \quad (47)$$

$$W_{ji}^k = \begin{bmatrix} 0_{q_{ji}^k \times n} & I_{q_{ji}^k} & 0_{q_{ji}^k \times 1} \end{bmatrix}, \quad K_{ji}^k = \begin{bmatrix} 0_{1 \times n} & 0_{1 \times q_{ji}^k} & 1 \end{bmatrix}. \quad (48)$$

Theorem 4: SDP-Based Verification of Quasi-Linear Graph Models. Consider a program \mathcal{P} , and its quasi-linear graph model $G(\mathcal{N}, \mathcal{E})$. For $j \in \mathcal{N}$, Let $\alpha_j \succ 0$ be a diagonal matrix specifying the overflow limit and let $\Lambda_j = \text{diag}\{\alpha_j^{-2}, -1\}$. Let $\theta_{j\emptyset}^k = 0$ for all $(\emptyset, j, k) \in \mathcal{E}$. An overflow runtime error does not occur during any execution of \mathcal{P} , if for all $(i, j, k) \in \mathcal{E}$ the following LMIs are satisfied:

$$\mathcal{F}^T P_j \mathcal{F} - \theta_{ji}^k \mathcal{L}^T P_i \mathcal{L} \preceq \mathcal{W}^T D \mathcal{W} - (\text{Tr}(D) + \mu_{ji}^k) \mathcal{K}^T \mathcal{K} + \text{He}(Y \mathcal{H} \mathcal{L} + Z \mathcal{G} \mathcal{L}) + \lambda_{ji}^k \quad (49)$$

$$\mathcal{F}^T \Lambda_j \mathcal{F}^T - \mathcal{L}^T P_i \mathcal{L} \preceq \mathcal{W}^T \tilde{D} \mathcal{W} - \text{Tr}(\tilde{D}) \mathcal{K}^T \mathcal{K} + \text{He}(\tilde{Y} \mathcal{H} \mathcal{L} + \tilde{Z} \mathcal{G} \mathcal{L}) + \tilde{\lambda}_{ji}^k \quad (50)$$

where for all $(i, j, k) \in \mathcal{E}$ we have

$$\lambda_{ji}^k = \sum_s \eta_{ji,s}^k \mathcal{L}^T Q_{ji,s}^k \mathcal{L} + \sum_m \rho_{ji,m}^k \mathcal{L}^T R_{ji,m}^k \mathcal{L} \quad (51)$$

$$\tilde{\lambda}_{ji}^k = \sum_s \tilde{\eta}_{ji,s}^k \mathcal{L}^T Q_{ji,s}^k \mathcal{L} + \sum_m \tilde{\rho}_{ji,m}^k \mathcal{L}^T R_{ji,m}^k \mathcal{L} \quad (52)$$

$$(Y, \tilde{Y}, Z, \tilde{Z}, D, \tilde{D}) = (Y_{ij}^k, \tilde{Y}_{ij}^k, Z_{ij}^k, \tilde{Z}_{ij}^k, D_{ij}^k, \tilde{D}_{ij}^k) \quad (53)$$

$$(\mathcal{F}, \mathcal{G}, \mathcal{H}, \mathcal{K}, \mathcal{L}, \mathcal{W}) = (F_{ji}^k, G_{ji}^k, H_{ji}^k, K_{ji}^k, L_{ji}^k, W_{ji}^k) \quad (54)$$

where the \mathcal{S} -Procedure multipliers in (51)–(53) are constrained as follows:

$$\eta_{ji,s}^k, \tilde{\eta}_{ji,s}^k \in \mathbb{R}_-, \quad \forall (i, j, k) \in \mathcal{E}, \quad s \in S_{ji}^k \quad (55)$$

$$\rho_{ji,m}^k, \tilde{\rho}_{ji,m}^k \in \mathbb{R}, \quad \forall (i, j, k) \in \mathcal{E}, \quad m \in M_{ji}^k \quad (56)$$

$$Y_{ji}^k, \tilde{Y}_{ji}^k \in \mathbb{R}^{(n+q_{ji}^k+1) \times h_{ji}^k}, \quad \forall (i, j, k) \in \mathcal{E} \quad (57)$$

$$Z_{ji}^k, \tilde{Z}_{ji}^k \in \mathbb{R}_-^{(n+q_{ji}^k+1) \times g_{ji}^k}, \quad \forall (i, j, k) \in \mathcal{E} \quad (58)$$

$$D_{ij}^k, \tilde{D}_{ij}^k \in \mathbb{D}_+^{q_{ji}^k \times q_{ji}^k}, \quad \forall (i, j, k) \in \mathcal{E}, \quad (59)$$

and the matrices in (54) are defined in (46)–(48). In addition, if for every simple cycle $\mathcal{C} \in G$, we have:

$$\theta(\mathcal{C}) + \mu(\mathcal{C}) > 1, \quad \mu(\mathcal{C}) > 0, \quad (60)$$

with $\theta(\mathcal{C})$ and $\mu(\mathcal{C})$ defined as in (28), then \mathcal{P} terminates in at most T_u iterations where

$$T_u = \sum_{\mathcal{C} \in G: \theta(\mathcal{C}) \neq 1} \frac{\log(\theta(\mathcal{C}) + \mu(\mathcal{C}) - 1) - \log \mu(\mathcal{C})}{\log \theta(\mathcal{C})} + \sum_{\mathcal{C} \in G: \theta(\mathcal{C}) = 1} \frac{1}{\mu(\mathcal{C})}.$$

Proof: See Appendix II. ■

Remarks

- 1) A simpler, computationally less demanding but possibly more conservative variation of Theorem 4 can be formulated by replacing (50) with

$$\Lambda_i - P_i \preceq 0, \quad \forall i \in \mathcal{N}. \quad (61)$$

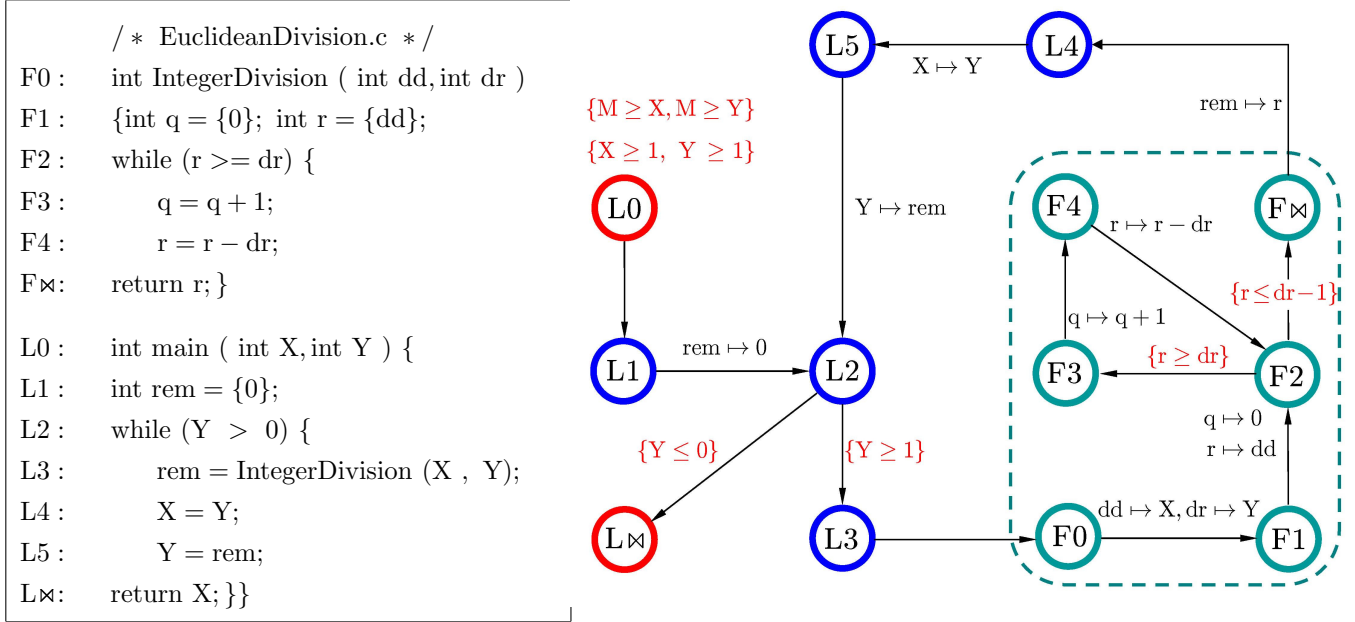
Subsequently, all the associated multipliers (i.e., the variables denoted by *tilde* characters in (55)–(59)) will be eliminated.

- 2) It is possible to use the same set of multipliers (i.e., the variables defined in (55)–(59)) across all edges $(i, j, k) \in \mathcal{E}$. This approach would also result in a computationally less demanding but possibly conservative formulation.
- 3) To improve the chances of finding stronger invariants, quadratic equality and inequality constraints can be generated from linear constraints and added to the set of quadratic constraints in characterization of the set $X_i \cap \Pi_{ji}^k$ (cf. equation 46). Given scalar linear constraints of the form $G_{ji,r}^k \bar{x} \geq 0$, $r \in \mathbb{Z}(1, g_{ji}^k)$ and/or $H_{ji,s}^k \bar{x} = 0$, $s \in \mathbb{Z}(1, h_{ji}^k)$, define $R_{s_1 s_2} = H_{s_1}^T H_{s_2} + H_{s_2}^T H_{s_1}$, $R_{s_1 r_1} = H_{s_1}^T G_{r_1} + G_{r_1}^T H_{s_1}$, and $Q_{r_1 r_2} = G_{r_1}^T G_{r_2} + G_{r_2}^T G_{r_1}$. Then, we have $\bar{x}^T R_{s_1 s_2} \bar{x} = 0$, $\bar{x}^T R_{s_1 r_1} \bar{x} = 0$, and $\bar{x}^T Q_{r_1 r_2} \bar{x} \geq 0$, for all $r_1, r_2 \in \mathbb{Z}(1, g_{ji}^k)$ and $s_1, s_2 \in \mathbb{Z}(1, h_{ji}^k)$.
- 4) Replacing (61) with $z\Lambda_i - P_i \preceq 0$, where $z > 0$ is a decision variable can improve the scaling and numerical conditioning of the associated semidefinite program, resulting in stronger invariants and improved analysis. The statement of the theorem on absence of overflow remains unchanged under this modification. Only the upper bound on the maximum number of iterations needs to be adjusted accordingly since we now have $\|\sigma(\mathcal{C})\|_\infty \leq z$ instead of $\|\sigma(\mathcal{C})\|_\infty \leq 1$. The same scaling arguments apply to (50).
- 5) MATLAB implementations of Theorem 4 and a few variations of it can be found in [65].

V. CASE STUDIES

A. Euclidean Division

In this section we apply the framework to the analysis of Program 4 displayed below.



Program 4: Euclidean Division and its Graph Model

Program 4 takes two positive integers $X \in [1, M]$ and $Y \in [1, M]$ as the input and returns their greatest common divisor by implementing the Euclidean Division algorithm. Note that the MAIN function in Program 4 uses the INTEGERDIVISION program (Program 1).

1) *Global Analysis:* A global model can be constructed by embedding the dynamics of the INTEGERDIVISION program within the dynamics of MAIN. A labeled graph model is shown alongside the text of the program. This model has a state space $X = \mathcal{N} \times [-M, M]^7$, where \mathcal{N} is the set of nodes as shown in the graph, and the global state $x = [X, Y, \text{rem}, \text{dd}, \text{dr}, q, r]$ is an element of the hypercube $[-M, M]^7$. A *reduced* graph model can be obtained by combining the effects of consecutive transitions and relabeling the reduced graph model accordingly. While analysis of the full graph model is possible, working with a *reduced* model is computationally advantageous. Furthermore, mapping the properties of the reduced graph model to the original model is algorithmic. Interested readers may consult [59] for further elaboration on this topic. For the graph model of Program 4, a reduced model can be obtained

The figure consists of two directed graphs. The left graph has four nodes: L0 (red circle), L2 (blue circle), F2 (teal circle), and L∞ (red circle). Edges are: L0 → L2, L2 → F2, L2 → L∞, and a self-loop on F2. A curved arrow also points from L2 to F2. The right graph has three nodes: L0 (red circle), F2 (teal circle), and L∞ (red circle). Edges are: L0 → F2, F2 → L∞, and two self-loops on F2. The top self-loop is labeled $\bar{T}_{F_2 F_2}^2$ and the bottom self-loop is labeled $\bar{T}_{F_2 F_2}^1$. Red text labels $\Pi_{F_2 F_2}^2$ and $\Pi_{F_2 F_2}^1$ are also present near the self-loops.

$$\begin{array}{ll} \overline{T}_{\mathbf{F}_2\mathbf{F}_2}^1 : x \mapsto [X, Y, \text{rem}, \text{dd}, \text{dr}, q+1, r-\text{dr}] & \overline{T}_{\mathbf{F}_2\mathbf{F}_2}^2 : x \mapsto [Y, r, r, Y, r, 0, Y] \\ \overline{T}_{\mathbf{L}_0\mathbf{F}_2} : x \mapsto [X, Y, 0, X, Y, 0, X] & \overline{T}_{\mathbf{F}_2\mathbf{L}_\infty} : x \mapsto [Y, r, r, \text{dd}, \text{dr}, q, r] \\ \Pi_{\mathbf{F}_2\mathbf{F}_2}^2 : \{x \mid 1 \leq r \leq \text{dr} - 1\} & \Pi_{\mathbf{F}_2\mathbf{F}_2}^1 : \{x \mid r \geq \text{dr}\} & \Pi_{\mathbf{F}_2\mathbf{L}_\infty} : \{x \mid r \leq \text{dr} - 1, r \leq 0\} \end{array}$$

We are interested in generating certificates of termination and absence of overflow. First, by recursively searching for linear invariants we are able to establish simple lower bounds on all variables in just two rounds (the properties established in the each round are added to the model and the next round of search begins). For instance, the property $X \geq 1$ is established only after $Y \geq 1$ is established. These results, which were obtained by applying the first part of Theorem 3 (equations (43)-(44) only) with linear functionals are summarized in Table II.

[illegible]

We then add these properties to the node invariant sets to obtain stronger invariants that certify FTT and boundedness of all variables in $[-M, M]$. By applying Theorem 3 and SOS programming using YALMIP [35], the following invariants are found² (after post-processing, rounding the coefficients, and reverifying):

$$\begin{aligned}\sigma_{1F_2}(x) &= 0.4(Y - M)(2 + M - r) & \sigma_{2F_2}(x) &= (q \times Y + r)^2 - M^2 \\ \sigma_{3F_2}(x) &= (q + r)^2 - M^2 & \sigma_{4F_2}(x) &= 0.1(Y - M + 5Y \times M + Y^2 - 6M^2) \\ \sigma_{5F_2}(x) &= Y + r - 2M + Y \times M - M^2 & \sigma_{6F_2}(x) &= r \times Y + Y - M^2 - M\end{aligned}$$

The properties proven by these invariants are summarized in the Table III. The specifications that the program terminates and that $x \in [-M, M]^7$ for all initial conditions $X, Y \in [1, M]$, could not be established in one shot, at least when trying polynomials of degree $d \leq 4$. For instance, σ_{1F_2} certifies boundedness of all the variables except q , while σ_{2F_2} and σ_{3F_2} which certify boundedness of all variables including q do not certify FTT. Furthermore, boundedness of some of the variables is established in round II, relying on boundedness properties proven in round I. Given $\sigma(x) \leq 0$ (which is found in round I), second round verification can be done by searching for a strictly positive polynomial $p(x)$ and a nonnegative polynomial $q(x) \geq 0$ satisfying:

$$q(x) \sigma(x) - p(x) ((\bar{T}x)_i^2 - M^2) \geq 0, \quad \bar{T} \in \{\bar{T}_{F_2F_2}^1, \bar{T}_{F_2F_2}^2\} \quad (62)$$

where the inequality (62) is further subject to boundedness properties established in round I, as well as the usual passport conditions and basic invariant set conditions.

TABLE III

Invariant $\sigma_{F_2}(x) =$	$\sigma_{1F_2}(x)$	$\sigma_{2F_2}(x), \sigma_{3F_2}(x)$	$\sigma_{4F_2}(x)$	$\sigma_{5F_2}(x), \sigma_{6F_2}(x)$
$(\theta_{F_2F_2}^1, \mu_{F_2F_2}^1)$	(1, 0)	(1, 0)	(1, 0)	(1, 1)
$(\theta_{F_2F_2}^2, \mu_{F_2F_2}^2)$	(1, 0.8)	(0, 0)	(1, 0.7)	(1, 1)
Round I: $x_i^2 \leq M^2$ for $x_i =$	Y, X, r, dr, rem, dd	q, Y, dr, rem	Y, X, r, dr, rem, dd	Y, dr, rem
Round II: $x_i^2 \leq M^2$ for $x_i =$		X, r, dd		X, r, dd
Certificate for FTT	NO	NO	NO	YES, $T_u = 2M^2$

²Different choices of polynomial degrees for the Lyapunov invariant function and the multipliers, as well as different choices for θ, μ and different rounding schemes lead to different invariants. Note that rounding is not essential.

In conclusion, $\sigma_{2F_2}(x)$ or $\sigma_{3F_2}(x)$ in conjunction with $\sigma_{5F_2}(x)$ or $\sigma_{6F_2}(x)$ prove finite-time termination of the algorithm, as well as boundedness of all variables within $[-M, M]$ for all initial conditions $X, Y \in [1, M]$, for any $M \geq 1$. The provable bound on the number of iterations certified by $\sigma_{5F_2}(x)$ and $\sigma_{6F_2}(x)$ is $T_u = 2M^2$ (Corollary 2). If we settle for more conservative specifications, e.g., $x \in [-kM, kM]^7$ for all initial conditions $X, Y \in [1, M]$ and sufficiently large k , then it is possible to prove the properties in one shot. We show this in the next section.

2) *MIL-GH Model*: For comparison, we also constructed the MIL-GH model associated with the reduced graph in Figure 1. The corresponding matrices are omitted for brevity, but details of the model along with executable Matlab verification codes can be found in [65]. The verification theorem used in this analysis is an extension of Theorem 2 to analysis of MIL-GHM for specific numerical values of M , though it is certainly possible to perform this modeling and analysis exercise for parametric bounded values of M . The analysis using the MIL-GHM is in general more conservative than SOS optimization over the graph model presented earlier. This can be attributed to the type of relaxations proposed (similar to those used in Lemma 1) for analysis of MILMs and MIL-GHMs. The benefits are simplified analysis at a typically much less computational cost. The certificate obtained in this way is a single quadratic function (for each numerical value of M), establishing a bound $\gamma(M)$ satisfying $\gamma(M) \geq (X^2 + Y^2 + \text{rem}^2 + \text{dd}^2 + \text{dr}^2 + \text{q}^2 + \text{r}^2)^{1/2}$. Table IV summarizes the results of this analysis which were performed using both Sedumi 1_3 and LMILAB solvers.

TABLE IV

M	10^2	10^3	10^4	10^5	10^6
Solver: LMILAB [22]: $\gamma(M)$	5.99M	6.34M	6.43M	6.49M	7.05M
Solver: SEDUMI [61]: $\gamma(M)$	6.00M	6.34M	6.44M	6.49M	NAN
$(\theta_{F2F2}^1, \mu_{F2F2}^1)$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$
$(\theta_{F2F2}^2, \mu_{F2F2}^2)$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$	$(1, 10^{-3})$
Upperbound on iterations	$T_u = 2e4$	$T_u = 8e4$	$T_u = 8e5$	$T_u = 1.5e7$	$T_u = 3e9$

3) *Modular Analysis*: The preceding results were obtained by analysis of a global model which was constructed by embedding the internal dynamics of the program's functions within the global dynamics of

the Main function. In contrast, the idea in *modular analysis* is to model software as the interconnection of the program's "building blocks" or "modules", i.e., functions that interact via a set of *global* variables. The dynamics of the functions are then abstracted via Input/Output behavioral models, typically constituting equality and/or inequality constraints relating the input and output variables. In our framework, the invariant sets of the terminal nodes of the modules (e.g., the set X_{∞} associated with the terminal node F_{∞} in Program 4) provide such I/O model. Thus, richer characterization of the invariant sets of the terminal nodes of the modules are desirable. Correctness of each sub-module must be established separately, while correctness of the entire program will be established by verifying the unreachability and termination properties w.r.t. the global variables, as well as verifying that a terminal global state will be reached in finite-time. This way, the program variables that are *private* to each function are abstracted away from the global dynamics. This approach has the potential to greatly simplify the analysis and improve the scalability of the proposed framework as analysis of large size computer programs is undertaken. In this section, we apply the framework to modular analysis of Program 4. Detailed analysis of the advantages in terms of improving scalability, and the limitations in terms of conservatism the analysis is an important and interesting direction of future research.

The first step is to establish correctness of the INTEGERDIVISION module, for which we obtain

$$\sigma_{7F2}(dd, dr, q, r) = (q + r)^2 - M^2$$

The function σ_{7F2} is a $(1, 0)$ -invariant proving boundedness of the state variables of INTEGERDIVISION.

Subject to boundedness, we obtain the function

$$\sigma_{8F2}(dd, dr, q, r) = 2r - 11q - 6Z$$

which is a $(1, 1)$ -invariant proving termination of INTEGERDIVISION. The invariant set of node F_{∞} can thus be characterized by

$$X_{\infty} = \{(dd, dr, q, r) \in [0, M]^4 \mid r \leq dr - 1\}$$

The next step is construction of a global model. Given X_{∞} , the assignment at L3:

$$L3 : \text{rem} = \text{IntegerDivision}(X, Y)$$

can be abstracted by

$$\text{rem} = W, \text{ s.t. } W \in [0, M], \quad W \leq Y - 1,$$

allowing for construction of a global model with variables X, Y , and rem , and an external state-dependent input $W \in [0, M]$, satisfying $W \leq Y - 1$. Finally, the last step is analysis of the global model. We obtain the function $\sigma_{9L2}(X, Y, \text{rem}) = Y \times M - M^2$, which is $(1, 1)$ -invariant proving both FTT and boundedness of all variables within $[M, M]$.

B. Filtering

Program 4 has been adopted with some minor modifications from a similar program analyzed by ASTREE [8], [66]. The only modification that we have made is the addition of the real-time input $w \in [-1, 1]$ in line L4. Note that when $B = 0$ (in line L4) we have the same program reported in [66]. In Program 4, the function $\text{saturate}(\cdot)$ truncates the real-time input $*\text{PtrToInput}$ so that $w \in [-1, 1]$ is guaranteed. We first build a graph model of this program. The variables of the graph model are:

$Z, Y, E[0], E[1], S[0], S[1], \text{INIT}$.

The variable INIT is Boolean which we model by $v \in \{-1, 1\}$ in the following way:

$$\text{INIT} = \text{True} \Leftrightarrow v = 1, \text{ and } \text{INIT} = \text{False} \Leftrightarrow v = -1.$$

The graph model of this program is shown in Figure 2.

```

/* filter.c */
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float Y={0}, Z={0};
F0 : void filter () {
F1 : static float E[2], S[2];
F2 : if (INIT) {
F3 :     S[0] = Z;
F4 :     Y = Z;
F5 :     E[0] = Z;
F6 : } else {
F7 :     Y = (((((0.5*Z) - (E[0]*0.7)) + (E[1]*0.4)) + (S[0]*1.5)) - (S[1]*0.7));
F8 : }
F9 : E[1] = E[0];
F10 : E[0] = Z;
F11 : S[1] = S[0];
F12 : S[0] = Y;
F13 : }

L0 : void main () {
L1 : Z = 0.2 * Z + 5;
L2 : INIT = TRUE;
L3 : while (1) {
        wait (0.001); w = saturate(*PtrToInput); /*updates real-time input*/
L4 :     Z = 0.9 * Z + 35+B*w;
L5 :     filter ();
L6 :     INIT = FALSE;
L7 : }
L∞: }

```

Program 4: Example of filtering in safety-critical software

Example adapted from reference [66] with minor modifications.

In order to construct a more compact model, several lines of code corresponding to consecutive transitions have been combined, and only the first line corresponding to a series of consecutive transitions has been assigned a node on the graph. Specifically, the combined lines are: {L1, L2}, and {F3, F4, F5}, and {F9, F10, F11, F12}. The net effect of the eliminated lines is captured in the model by taking the composition of the functions that are coded at these lines.

obtain a certificate for $M \geq \|x\|_2$ using one quadratic invariant. In Method II, for each $i \in \{1, \dots, 7\}$ we set $\Lambda_j = \Lambda = \text{diag} \{M^{-2}e_i, -1\}$, where e_i is the i -th standard unit vector in \mathbb{R}^7 (cf. Corollary 1 and Equation (23)). As a result, we obtain a much tighter upperbound in the form of $M \geq \|x\|_\infty$ at the expense of more computations. Table V summarizes the results of this analysis. Analysis of the floating-point operations is based on the abstractions discussed in Appendix I. The corresponding Matlab codes can be found in [65]. In this analysis, the solver SeDuMi version 1.3 [61] is used for semidefinite optimization, and Yalmip [35] for compilation. For comparison we point out that the static analyzer Astree reports a bound of $M = 1327.05$ for the $B = 0$ case [66].

TABLE V

	Computations:	infinite-precision	64-bit double precision	32-bit single precision	
Method I	$B = 0$	$M = 884.95$	$M = 884.96$	$M = 884.96$	$M \geq \ x\ _2 \geq \ x\ _\infty$
Method II	$B = 0$	$M = 373.11$	$M = 373.12$	$M = 373.12$	$M \geq \ x\ _\infty$
Method I	$B = 20$	$M = 1400.95$	$M = 1402.81$	$M = 1402.87$	$M \geq \ x\ _2 \geq \ x\ _\infty$
Method II	$B = 20$	$M = 609.83$	$M = 609.83$	$M = 609.83$	$M \geq \ x\ _\infty$
	(θ_{ij}, μ_{ij})	$(0.98, 0)$	$(0.98, 0)$	$(0.98, 0)$	

VI. CONCLUDING REMARKS

We took a systems-theoretic approach to software analysis, and presented a framework based on convex optimization of Lyapunov invariants for verification of a range of important specifications for software systems, including finite-time termination and absence of run-time errors such as overflow, out-of-bounds array indexing, division-by-zero, and user-defined program assertions. The verification problem is reduced to solving a numerical optimization problem, which when feasible, results in a certificate for the desired specification. The novelty of the framework, and consequently, the main contributions of this paper are in the systematic transfer of Lyapunov functions and the associated computational techniques from control systems to software analysis. The presented work can be extended in several directions. These include understanding the limitations of modular analysis of programs, perturbation analysis of the Lyapunov certificates to quantify robustness with respect to round-off errors, extension to systems with software in closed loop with hardware, and adaptation of the framework to specific classes of software.

VII. APPENDIX I

A. Semialgebraic Set-Valued Abstractions of Commonly-Used Nonlinearities

– Trigonometric Functions:

Abstraction of trigonometric functions can be obtained by approximation of the Taylor series expansion followed by representation of the absolute error by a static bounded uncertainty. For instance, an abstraction of the $\sin(\cdot)$ function can be constructed as follows:

Abstraction of $\sin(x)$	$x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$	$x \in [-\pi, \pi]$
$\overline{\sin}(x) \in \{x + aw \mid w \in [-1, 1]\}$	$a = 0.571$	$a = 3.142$
$\overline{\sin}(x) \in \{x - \frac{1}{6}x^3 + aw \mid w \in [-1, 1]\}$	$a = 0.076$	$a = 2.027$

Abstraction of $\cos(\cdot)$ is similar. It is also possible to obtain piecewise linear abstractions by first approximating the function by a piece-wise linear (PWL) function and then representing the absolute error by a bounded uncertainty. Section II-B (Proposition 1) establishes universality of representation of generic PWL functions via binary and continuous variables and an algorithmic construction is given in the proof of the proposition in Appendix II. For instance, if $x \in [0, \pi/2]$ then a piecewise linear approximation with absolute error less than 0.06 can be constructed in the following way:

$$S = \{(x, v, w) \mid x = 0.2[(1+v)(1+w_2) + (1-v)(3+w_2)], (w, v) \in [-1, 1]^2 \times \{-1, 1\}\} \quad (63a)$$

$$\overline{\sin}(x) \in \{Tx_E \mid x_E \in S\}, \quad T : x_E \mapsto 0.45(1+v)x + (1-v)(0.2x + 0.2) + 0.06w_1 \quad (63b)$$

– The Sign Function (sgn) and the Absolute Value Function (abs):

The sign function ($\text{sgn}(x) = 1\mathbb{I}_{[0, \infty)}(x) - 1\mathbb{I}_{(-\infty, 0)}(x)$) may appear explicitly as one of the program's functions, or implicitly as a model for conditional switching. A particular abstraction of $\text{sgn}(\cdot)$ is as follows: $\overline{\text{sgn}}(x) \in \{v \mid xv \geq 0, v \in \{-1, 1\}\}$. Note that $\text{sgn}(0)$ is equal to 1, while the abstraction is multi-valued at zero: $\overline{\text{sgn}}(0) \in \{-1, 1\}$. The absolute value function can be represented (precisely) over $[-1, 1]$ in the following way:

$$\text{abs}(x) = \{xv \mid x = 0.5(v + w), (w, v) \in [-1, 1] \times \{-1, 1\}\}$$

– Modulo Arithmetic:

Consider the function $\text{mod} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ defined in the following way:

$$\text{mod}(t, s) = t - ns, \text{ where } n = \lfloor \frac{t}{s} \rfloor.$$

Abstraction of the function $\text{mod}(\cdot, \cdot)$ over bounded subsets of the set of integers is as follows. Let

$t \in [0, Ms)$, where $M > 0$ is an integer. Then:

$$\text{mod}(t, s) \in \left\{ t - \frac{1}{2}s \sum_{k=1}^{M-1} (1 + v_k) \mid (t - ks)v_k \geq 0, v_k \in \{-1, 1\}, k = 1, \dots, M-1 \right\}. \quad (64)$$

It follows from (64) that the result of summation modulo s of two integers $t_1, t_2 \in [0, Ms)$ can be abstracted in the following way:

$$\text{mod}(t_1 + t_2, s) \in \left\{ t_1 + t_2 - \frac{1}{2}s \sum_{k=1}^{2M-1} (1 + v_k) \mid (t_1 + t_2 - ks)v_k \geq 0, v_k \in \{-1, 1\}, k = 1, \dots, 2M-1 \right\}.$$

B. Floating-Point and Fixed-Point Arithmetic

This subsection is based on [42], Chapter 7. Interested readers are referred to [42], [43] for a more comprehensive discussion of computations with floats. For computations with floating-point numbers, the IEEE 754-1985 norm has become the hardware standard, and is supported by most popular programming languages such as C. In this standard, a *float* number is represented by a triplet (s, f, e) , where:

- $s \in \{0, 1\}$ is the sign bit.
- f is the fractional part, represented by a p -bit unsigned integer: $f_1 \dots f_p, f_i \in \{0, 1\}$.
- e is the biased exponent, represented by a q -bit unsigned integer: $e_1 \dots e_q, e_i \in \{0, 1\}$.

A floating point number $z = (s, f, e)$ is then in one of the following forms:

- $z = (-1)^s \times 2^{e-bias} \times 1.f$, when $1 \leq e \leq 2^q - 2$.
- $z = (-1)^s \times 2^{1-bias} \times 0.f$, when $e = 0$, and $f \neq 0$.
- $z = +0$ (when $s = 1$) and $z = -0$ (when $s = 0$) and $e = f = 0$.
- $z = +\infty$ (when $s = 1$) and $z = -\infty$ (when $s = 0$) and $e = 2^q - 1, f \neq 0$.
- $z = NaN$ when $e = 2^q - 1, f = 0$.

The values of p, q , and *bias* depend on the specific format:

- If format is 32-bit single precision (f=32), then $bias = 127, q = 8$, and $p = 23$.
- If format is 64-bit double precision (f=64), then $bias = 1023, q = 11$, and $p = 52$.

Other formatting standards such as *long double* or *quadruple* precision also exist. In floating point computations, the result of performing the arithmetic operation $\otimes \in \{+, -, \times, \div\}$ on two *float* variables x and y is stored in a *float* variable $z := \text{float}(x \otimes y, f)$ which is a complicated function of x , y , and the format f . Examples of the format f include the IEEE 754 with 32-bit single precision, or IEEE 754 with 64-bit double precision. A floating-point operation is equivalent to performing the operation over the set of real numbers followed by rounding the result to a *float* [43], [42]. In IEEE 754 the possible rounding modes are rounding towards 0, towards $+\infty$, towards $-\infty$, and to the nearest (n). The rounding function $\Gamma_{f,m} : \mathbb{R} \rightarrow \mathbb{F} \cup \{\epsilon\}$ maps a real number to a float number or to runtime error ϵ , depending on the format ‘f’ and the rounding mode ‘m’. We refer the reader to [42] for more details on the rounding function. For our purposes, it is sufficient to say that for all rounding modes, the following relation holds:

$$\forall x \in [-\alpha_f, \alpha_f] : |\Gamma_{f,m}(x) - x| \leq \gamma_f |x| + \beta_f$$

where $\gamma_f := 2^{-p}$, and $\alpha_f := (2 - 2^{-p}) 2^{2^q - \text{bias} - 2}$ is the largest non-infinite number, and $\beta_f := 2^{1 - \text{bias} - p}$ is the smallest non-zero positive number.

Based on the above discussions, an abstraction of the floating-point arithmetic operators can be constructed in the following way:

$$x + y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x + y) = z \in \{x + y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| + |y|) + \beta_f\}$$

$$x - y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x - y) = z \in \{x - y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| + |y|) + \beta_f\}$$

$$x \times y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x \times y) = z \in \{x \times y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| \times |y|) + \beta_f\}$$

$$x \div y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x \div y) = z \in \{x \div y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| \div |y|) + \beta_f\}$$

where the constants α_f , β_f and γ_f are defined as before. The above abstractions can still be complicated as the magnitude of δ depends on the operands x and y . If all of the program variables (including the result of the arithmetic operation) reside in $[-\alpha, \alpha]$ where $\alpha \ll \alpha_f$ then a simpler but more conservative abstraction can be constructed in the following way:

$$x \otimes y \in [-\alpha, \alpha] \Rightarrow \text{float}(x \otimes y) = z \in \{x \otimes y + \delta w \mid w \in [-1, 1], \delta = \alpha \gamma_f + \beta_f\}$$

For instance, if $f=32$ and $\alpha = 10^6$, then $\delta = 0.12$, and if $f=64$ and $\alpha = 10^{10}$, then $\delta = 2.3 \times 10^{-6}$.

Abstractions of arithmetic operations in fixed-point computations is similar to the above. The magnitude of δ will depend on the number of bits and the dynamic range. For instance, in the two's complement format we have: $\delta = \rho (2^b - 1)^{-1}$, where ρ is the dynamic range and b is the number of bits. A case study in software verification based on the above discussed abstractions of floating-point computations has been presented in Section V-B.

C. Graph models with state-dependent or time-varying edges

Consider the following fragment from a program with two variables: $x \in \mathbb{R}$, and $i \in \{1, 2\}$:

L1 : $x = A[i] x$;
L2 : expression

Then, the state transition from node 1 to 2 is defined by $T_{21} : (x, i) \rightarrow (A[i] x, i)$, where A is an array of size 2. An algorithm similar to the one used in construction of MILMs (cf. Proposition 1) can be used to construct a transition label using semi-algebraic set-valued maps. For the above example, this can be done in the following way:

$$\begin{aligned} \overline{T}_{21}(x, i) &= \{T_{21}(x, i, v_1, v_2) \mid (x, i, v_1, v_2) \in S_{21}\}. \\ T_{21}(x, i, v_1, v_2) &= A[1] v_1 x + A[2] v_2 x. \\ S_{21} &= \{(x, i, v_1, v_2) \mid v_1 + v_2 = 1, v_1, v_2 \in \{0, 1\}, v_1(i-1) + v_2(i-2) = 0\}. \end{aligned}$$

In light of Proposition 1, generalization of this technique to multidimensional arrays with arbitrary finite size is straightforward. However, the number of binary decision variables grows (linearly) with the number of elements in the array, which can be undesirable for large arrays. An alternative approach to constructing graph models with fixed labels is to attempt to derive a fixed map by computing the net effect of several lines of code [21]. When applicable, the result is a fixed map which is obtained by taking the composition of several functions. This is an instance of a more general concept in computer science, namely, extracting the higher level semantics, which allows one to define more compact models of the software.

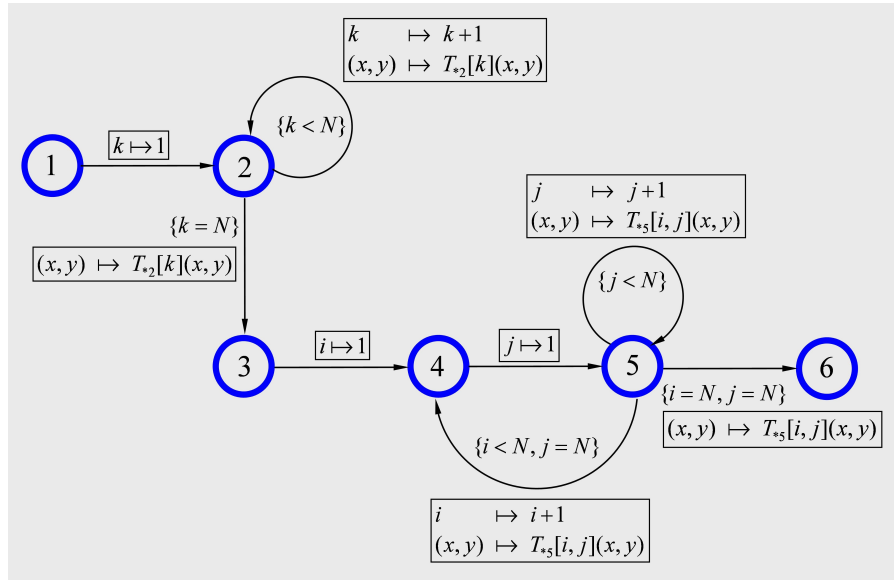


Fig. 3. Graph model of a code fragment (Program 5) with state-dependent labels. The transition labels are shown in boxes and the passport labels are in brackets. For simplicity, only the non-identity transition labels are shown.

For instance, consider the following code fragment:

```

L1 :   for ( k = 1 ; k == N ; k++)   {
L2 :     y[ k ] = x[ k ] ; x[ k ] = 0 ; }
L3 :   for ( i = 1 ; i == N ; i++)   {
L4 :     for ( j = 1 ; j == N ; j++)   {
L5 :       x[ i ] = x[ i ] + y[ j ] * A[ i ][ j ] ;
L6 :     }}

```

Program 5: A code fragment from a linear filtering application

A graph model of this code fragment is shown in Figure 3, where:

$$y = [y[1] \cdots y[N]]^T, \quad x = [x[1] \cdots x[N]]^T$$

$$T_{*2}[k] : \begin{bmatrix} y \\ x \end{bmatrix} \rightarrow \begin{bmatrix} I - e_{kk} & e_{kk} \\ 0 & I - e_{kk} \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix},$$

$$T_{*5}[i, j] : \begin{bmatrix} y \\ x \end{bmatrix} \rightarrow \begin{bmatrix} I & 0 \\ A_{ij}e_{ij} & I \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix},$$

where e_{ij} is an $N \times N$ matrix which is zero everywhere except at the (i, j) -th entry which is 1, and A_{ij} denotes the (i, j) -th entry of A ($A_{ij} \equiv A[i][j]$) which is an $N \times N$ array in the code [21]. The remaining transition labels correspond to the counter variables i, j, k , and have been specified on the graph in Figure 3. From node 1 to node 3, the net effect is:

$$\prod_{k=1}^N T_{*2}[k] = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix},$$

and from node 3 to node 6 the net effect is:

$$\prod_{i=1}^N \prod_{j=1}^N T_{*5}[i, j] = \begin{bmatrix} I & 0 \\ A & 0 \end{bmatrix}.$$

VIII. APPENDIX II

Proof of Proposition 1: The proof is by construction. Let $X = \bigcup_{i=1}^N X_i$, where the X_i 's are compact polytopic sets. Further, assume that each X_i is characterized by a finite set of linear inequality constraints:

$X_i = \{x \mid \mathbf{S}_i x \leq \mathbf{s}_i, \mathbf{S}_i \in \mathbb{R}^{N_i \times n}, \mathbf{s}_i \in \mathbb{R}^{N_i}\}$. Let $v \in \{-1, 1\}^N$, and consider the following sets:

$$G_{xv} \triangleq \left\{ (x, v) \mid \sum_{i=1}^N v_i = -N + 2, (\mathbf{S}_i x - \mathbf{s}_i)(v_i + 1) \leq 0, v_i \in \{-1, 1\} : i \in \mathbb{Z}(1, N) \right\},$$

$$G_{xy} \triangleq \left\{ (x, y) \mid \sum_{i=1}^N (1 + v_i)(A_i x + B_i) = y, (x, v) \in G_{xv} \right\}.$$

First, we prove that $G_{xy} = G_1$. Define N binary vectors: $\eta^j \in \{-1, 1\}^N$, $j \in \mathbb{Z}(1, N)$, according to the following rule: $\eta_i^j = 1$ if and only if $i = j$. Also define $\mathbb{I}(x) \stackrel{\text{def}}{=} \{j \in \mathbb{Z}(1, N) \mid x \in X_j\}$. Now, let $(x_0, f(x_0)) \in G_1$. Then

$$(x_0, \eta^j) \in G_{xv} : \forall j \in \mathbb{I}(x_0).$$

Therefore, $(x_0, 2A_j x_0 + 2B_j) \in G_{xy} : \forall j \in \mathbb{I}(x_0)$, which by supposition, implies that $(x_0, f(x_0)) \in G_{xy}$.

This proves that $G_1 \subseteq G_{xy}$. Now, let $(x_0, y_0) \in G_{xy}$. Then, there exists $\hat{v} \in \{-1, 1\}^N$, such that $(x_0, \hat{v}) \in G_{xv}$. Hence, there exists $j \in \mathbb{Z}(1, N)$, such that $x_0 \in X_j$, and $y_0 = 2A_j x_0 + 2B_j$. It follows from the definition of f that $(x_0, y_0) \in G_1$. This proves that $G_{xy} \subseteq G_1$. We have shown that $G_1 = G_{xy}$.

It remains to show that G_{xy} has a representation equivalent to G_2 .

Define new variables $u_i = xv_i \in [-1, 1]^n$. Then

$$G_{xy} \triangleq \left\{ (x, y) \mid y = \sum_{i=1}^N (A_i x + A_i u_i + B_i + B_i v_i), \mathbf{S}_i u_i + \mathbf{S}_i x - \mathbf{s}_i v_i - \mathbf{s}_i \leq 0 \right. \\ \left. \sum_{i=1}^N v_i = -N + 2, u_i = xv_i, v_i \in \{-1, 1\} : i \in \mathbb{Z}(1, N) \right\}. \quad (65)$$

The nonlinear map $(x, v_i) \mapsto u_i$ can be represented by an affine transformation involving auxiliary variables $\underline{z}_i \in [-1, 1]^n$, and $\bar{z}_i \in [-1, 1]^n$, subject to a set of linear constraints, in the following way:

$$u_i = 2\underline{z}_i - x - v_i \mathbf{1}_n + \mathbf{1}_n, \quad \underline{z}_i \leq v_i \mathbf{1}_n, \quad \bar{z}_i \leq -v_i \mathbf{1}_n, \quad \underline{z}_i = x - \bar{z}_i - \mathbf{1}_n \quad (66)$$

equivalently:

$$u_i = 2\underline{z}_i - x - (v_i - 1) \mathbf{1}_n, \quad \underline{z}_i = x - \bar{z}_i - \mathbf{1}_n, \quad \underline{z}_i = (v_i - 1) \mathbf{1}_n + \underline{w}_i, \quad \bar{z}_i = (-v_i - 1) \mathbf{1}_n + \bar{w}_i,$$

where $\underline{w}_i, \bar{w}_i \in [-1, 1]^n$. Since by assumption each X_i is bounded, for all $i \in \mathbb{Z}(1, N)$ and all $j \in \mathbb{Z}(1, N_i)$, the quantities

$$\underline{R}_{ij} = \min_{x \in X_i} \mathbf{S}_{ij} x - \mathbf{s}_{ij} \quad (67)$$

exist, are finite and can be computed by solving $N_i \times N$ linear programs given in (67) (\mathbf{S}_{ij} and \mathbf{s}_{ij} denote the j -th row of \mathbf{S}_i and \mathbf{s}_i respectively). Let $\underline{R}_i = \text{diag}_{j \in \mathbb{Z}(1, N_i)} \{\underline{R}_{ij}\}$. Then G_{xy} as defined in (65) is equivalent to:

$$G_{xy} \triangleq \left\{ (x, y) \mid y = \sum_{i=1}^N (A_i x + A_i u_i + B_i + B_i v_i), (x, u_i, v_i) \in \mathbf{H} \right\} \quad (68)$$

where $\mathbf{H} = \{(x, u_i, v_i) \mid x, u_i, v_i \text{ satisfy (69) for some } \underline{z}_i, \bar{z}_i, \underline{w}_i, \bar{w}_i \in [-1, 1]^n, \text{ and } w_i \in [-1, 1]^{N_i}\} :$

$$\mathbf{H} = \left\{ \begin{array}{ll} 0 = \mathbf{S}_i u_i + \mathbf{S}_i x - \mathbf{s}_i v_i - \mathbf{s}_i - \underline{R}_i (w_i + \mathbf{1}_{N_i}) & \underline{z}_i = x - \bar{z}_i - \mathbf{1}_n, \quad \underline{z}_i, \bar{z}_i \in [-1, 1]^n \\ u_i = 2\underline{z}_i - x - (v_i - 1) \mathbf{1}_n & \bar{z}_i = (-v_i - 1) \mathbf{1}_n + \bar{w}_i, \\ \underline{z}_i = (v_i - 1) \mathbf{1}_n + \underline{w}_i & \sum_{i=1}^N v_i = -N + 2, \quad v_i \in \{-1, 1\} \end{array} \right\} \quad (69)$$

The equality constraints that define the Matrix H are precisely the equalities in (69), while the equality constraint in (68) defines the Matrix F . This completes the proof. \blacksquare

Proof of Proposition 2: First, consider the unreachability property. Assume to the contrary, that \mathcal{P} does not satisfy unreachability w.r.t. X_- . Then, there exists a solution $\mathcal{X} \equiv x(\cdot)$ of $\mathcal{S}(X, f, X_0, X_\infty)$, and a positive integer $t_- \in \mathbb{Z}_+$ such that $x(0) \in X_0$, and $x(t_-) \in X_-$. It follows from $X_0 \subseteq \bar{X}_0$, and

$f(x) \subseteq \bar{f}(x)$ that $\mathcal{X} \equiv x(\cdot)$ is also a solution of $\bar{\mathcal{S}}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$. Therefore, we have:

$$x(t_-) \in X_- , X_- \subseteq \bar{X}_- \Rightarrow x(t_-) \in \bar{X}_-.$$

However, $x(t_-) \in \bar{X}_-$ contradicts the fact that $\bar{\mathcal{S}}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$ satisfies safety w.r.t. \bar{X}_- . Proof of FTT is similar: let $\mathcal{X} \equiv x(\cdot)$ be any solution of $\mathcal{S}(X, f, X_0, X_\infty)$. Since $\mathcal{X} \equiv x(\cdot)$ is also a solution of $\bar{\mathcal{S}}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$, it follows that there exists $t_T \in \mathbb{Z}_+$ such that $x(t_T) \in \bar{X}_\infty$. Since $x(t_T)$ is also an element of X , it follows that $x(t_T) \in \bar{X}_\infty \cap X$. Since by definition $\bar{X}_\infty \cap X \subset X_\infty$ holds, we must have $x(t_T) \in X_\infty$. ■

Proof of Proposition 4: Note that (13)–(15) imply that V is negative-definite along the trajectories of \mathcal{S} , except possibly for $V(x(0))$ which can be zero when $\eta = 0$. Let \mathcal{X} be any solution of \mathcal{S} . Since V is uniformly bounded on X , we have:

$$-\|V\|_\infty \leq V(x(t)) < 0, \forall x(t) \in \mathcal{X}, t > 1.$$

Now, assume that there exists a sequence $\mathcal{X} \equiv (x(0), x(1), \dots, x(t), \dots)$ of elements from X satisfying (1), but not reaching a terminal state in finite time. That is, $x(t) \notin X_\infty, \forall t \in \mathbb{Z}_+$. Then, it can be verified that if $t > T_u$, where T_u is given by (16), we must have: $V(x(t)) < -\|V\|_\infty$, which contradicts boundedness of V . ■

Proof of Theorem 1: Assume that \mathcal{S} has a solution $\mathcal{X} = (x(0), \dots, x(t_-), \dots)$, where $x(0) \in X_0$ and $x(t_-) \in X_-$. Let

$$\gamma_h = \inf_{x \in h^{-1}(X_-)} V(x)$$

First, we claim that $\gamma_h \leq \max\{V(x(t_-)), V(x(t_- - 1))\}$. If $h = I$, we have $x(t_-) \in h^{-1}(X_-)$ and $\gamma_h \leq V(x(t_-))$. If $h = f$, we have $x(t_- - 1) \in h^{-1}(X_-)$ and $\gamma_h \leq V(x(t_- - 1))$, hence the claim. Now, consider the $\theta = 1$ case. Since V is monotonically decreasing along solutions of \mathcal{S} , we must have:

$$\gamma_h = \inf_{x \in h^{-1}(X_-)} V(x) \leq \max\{V(x(t_-)), V(x(t_- - 1))\} \leq V(x(0)) \leq \sup_{x \in X_0} V(x) \quad (70)$$

which contradicts (17). Note that if $\mu > 0$ and $h = I$, then (70) holds as a strict inequality and we can

replace (17) with its non-strict version. Next, consider case (I), for which, V need not be monotonic along the trajectories. Partition X_0 into two subsets \overline{X}_0 and \underline{X}_0 such that $X_0 = \overline{X}_0 \cup \underline{X}_0$ and

$$V(x) \leq 0 \quad \forall x \in \underline{X}_0, \quad \text{and} \quad V(x) > 0 \quad \forall x \in \overline{X}_0$$

Now, assume that \mathcal{S} has a solution $\overline{\mathcal{X}} = (\overline{x}(0), \dots, \overline{x}(t_-), \dots)$, where $\overline{x}(0) \in \overline{X}_0$ and $\overline{x}(t_-) \in X_-$. Since $V(x(0)) > 0$ and $\theta < 1$, we have $V(x(t)) < V(x(0))$, $\forall t > 0$. Therefore,

$$\gamma_h = \inf_{x \in h^{-1}(X_-)} V(x) \leq \max\{V(x(t_-)), V(x(t_- - 1))\} \leq V(\overline{x}(0)) \leq \sup_{x \in X_0} V(x)$$

which contradicts (17). Next, assume that \mathcal{S} has a solution $\underline{\mathcal{X}} = (\underline{x}(0), \dots, \underline{x}(t_-), \dots)$, where $\underline{x}(0) \in \underline{X}_0$ and $\underline{x}(t_-) \in X_-$. In this case, regardless of the value of θ , we must have $V(\underline{x}(t)) \leq 0$, $\forall t$, implying that $\gamma_h \leq 0$, and hence, contradicting (18). Note that if $h = I$ and either $\mu > 0$, or $\theta > 0$, then (18) can be replaced with its non-strict version. Finally, consider case (II). Due to (19), V is strictly monotonically decreasing along the solutions of \mathcal{S} . The rest of the argument is similar to the $\theta = 1$ case. ■

Proof of Corollary 1: It follows from (21) and the definition of X_- that:

$$V(x) \geq \sup \left\{ \|\alpha^{-1}h(x)\|_q - 1 \right\} \geq \sup \left\{ \|\alpha^{-1}h(x)\|_\infty - 1 \right\} > 0, \quad \forall x \in X. \quad (71)$$

It then follows from (71) and (20) that:

$$\inf_{x \in h^{-1}(X_-)} V(x) > 0 \geq \sup_{x \in X_0} V(x)$$

Hence, the first statement of the Corollary follows from Theorem 1. The upperbound on the number of iterations follows from Proposition 4 and the fact that $\sup_{x \in X \setminus \{X_- \cup X_\infty\}} |V(x)| \leq 1$. ■

Proof of Corollary 2: The unreachability property follows directly from Theorem 1. The finite time termination property holds because it follows from (12), (24) and (27) along with Proposition 4, that the maximum number of iterations around every simple cycle \mathcal{C} is finite. The upperbound on the number of iterations is the sum of the maximum number of iterations over every simple cycle. ■

Proof of Lemma 1: Define $x_e = (x, w, v, 1)^T$, where $x \in [-1, 1]^n$, $w \in [-1, 1]^{n_w}$, $v \in \{-1, 1\}^{n_v}$.

Recall that $(x, 1)^T = L_2 x_e$, and that for all x_e satisfying $Hx_e = 0$, there holds: $(x_+, 1) = (Fx_e, 1) = L_1 x_e$.

It follows from Proposition 3 that (9) holds if:

$$x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e \leq -\mu, \text{ s.t. } H x_e = 0, L_3 x_e \in [-1, 1]^{n+n_w}, L_4 x_e \in \{-1, 1\}^{n_v}. \quad (72)$$

Recall from the \mathcal{S} -Procedure ((31) and (32)) that the assertion $\sigma(y) \leq 0, \forall y \in [-1, 1]^n$ holds if there exist nonnegative constants $\tau_i \geq 0, i = 1, \dots, n$, such that $\sigma(y) \leq \sum \tau_i (y_i^2 - 1) = y^T \tau y - \text{Trace}(\tau)$, where $\tau = \text{diag}\{\tau_i\} \succeq 0$. Similarly, the assertion $\sigma(y) \leq 0, \forall y \in \{-1, 1\}^n$ holds if there exist constants ρ_i such that $\sigma(y) \leq \sum \rho_i (y_i^2 - 1) = y^T \rho y - \text{Trace}(\rho)$, where $\rho = \text{diag}\{\rho_i\}$. Applying these relaxations to (72), we obtain sufficient conditions for (72) to hold:

$$x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e \leq x_e^T (Y H + H^T Y^T) x_e + x_e^T L_3^T D_{xw} L_3 x_e + x_e^T L_4^T D_v L_4 x_e - \mu - \text{Trace}(D_{xw} + D_v)$$

Together with $0 \preceq D_{xw}$, the above condition is equivalent to the LMIs in Lemma 1. ■

Proof of Theorem 4: Since $\theta_{j\emptyset}^k = 0$ for all $(\emptyset, j, k) \in \mathcal{E}$, we have $\sigma_j(x) \leq 0$, for all j in the outgoing set of node \emptyset . This replaces condition (24) in Corollary 2. The first part of the theorem thus follows from Corollary 2 and an application of the \mathcal{S} -Procedure relaxation technique in a similar fashion to the proof of Lemma 1. The second part of the theorem also follows from Corollary 2 by noting that due to (50), we have $\|\sigma(\mathcal{C})\|_\infty \leq 1$ for every simple cycle $\mathcal{C} \in G$. ■

REFERENCES

- [1] A. Adge. Optimisation et jeux appliqués à l'analyse statique de programmes par interprétation abstraite. Ph.D. Thesis, Ecole Polytechnique, France, 2011.
- [2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho X. Nicollin, A. Oliviero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems, *Theoretical Computer Science*, vol. 138, pp. 3–34, 1995.
- [3] R. Alur, T. Dang, and F. Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control*. LNCS v. 2289, pp. 35–48. Springer Verlag, 2002.
- [4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Soft. Eng.*, 29(6):524–541, 2003.
- [5] A. Bemporad, and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.
- [6] A. Bemporad, F. D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. LNCS v. 1790, pp. 45–58. Springer-Verlag, 2000.
- [7] D. Bertsimas, and J. Tsitsikilis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
- [8] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. LNCS v. 2566, pp. 85–108, Springer-Verlag, 2002.
- [9] J. Bochnak, M. Coste, and M. F. Roy. *Real Algebraic Geometry*. Springer, 1998.
- [10] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in Systems and Control Theory*, SIAM, 1994.
- [11] M. S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Trans. Aut. Ctrl.*, 43(4):475–482, 1998.
- [12] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: model and optimal control theory. *IEEE Trans. Aut. Ctrl.*, 43(1):31–45, 1998.
- [13] R. W. Brockett. Hybrid models for motion control systems. *Essays in Control: Perspectives in the Theory and its Applications*, Birkhauser, 1994.
- [14] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K.L. McMillan, and L.A. Ness. Verification of the Future-bus+cache coherence protocol. In *Formal Methods in System Design*, 6(2):217–232, 1995.
- [15] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [16] P. Cousot, and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.
- [17] P. Cousot. Abstract interpretation based formal methods and future challenges. LNCS, v. 2000:138–143, Springer, 2001.
- [18] P. Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In *Verification, Model Checking, and Abstract Interpretation 6th International Conference, VMCAI 2005*. Proceedings in Lecture Notes in Computer Science v. 3385, 2005.

- [19] D. Dams. Abstract Interpretation and Partition Refinement for Model Checking. Ph.D. Thesis, Eindhoven University of Technology, 1996.
- [20] E. Feron. Abstraction mechanisms across the board: A short introduction. In *A Workshop on Robustness, Abstractions and Computations*, Philadelphia, March 18, 2004. Presentations available at: http://web.mit.edu/feron/Public/wkshop_pres
- [21] E. Feron, and M. Roozbehani. Certifying controls and systems software. In *Proc. of the AIAA Guidance, Navigation and Control Conference*, Hilton Head, South Carolina, August 2007.
- [22] P. Gahinet, A. Nemirovskii, and A. Laub. LMILAB: A Package for Manipulating and Solving LMIs. South Natick, MA: The Mathworks, 1994.
- [23] ILOG Inc. ILOG CPLEX 9.0 User's guide. Mountain View, CA, 2003.
- [24] A. Girard, and G. J. Pappas. Verification using simulation. LNCS, v. 3927, pp. 272–286, Springer, 2006.
- [25] M. X. Goemans, and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery (ACM)*, 42(6):1115–1145, 1995.
- [26] S. V. Gusev, and A. L. Likhtarnikov. Kalman–Popov–Yakubovich Lemma and the \mathcal{S} -procedure: A historical essay. *Journal of Automation and Remote Control*, 67(11):1768–1810, 2006.
- [27] B. S. Heck, L. M. Wills, and G. J. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine*, 23(1):21–35, 2003.
- [28] M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier Science, 1977.
- [29] M. Johansson, and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Tran. Aut. Ctrl.* 43(4):555–559, 1998.
- [30] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.
- [31] H. Kopetz. *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer, 2001.
- [32] A. B. Kurzhanski, and I. Valyi. *Ellipsoidal Calculus for Estimation and Control*. Birkhauser, 1996.
- [33] G. Lafferriere, G. J. Pappas, and S. Sastry. Hybrid systems with finite bisimulations. LNCS, v. 1567, pp. 186–203, Springer, 1999.
- [34] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computations for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.
- [35] J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proc. of the CACSD Conference*, 2004. URL: <http://control.ee.ethz.ch/~joloef/yalmip.php>
- [36] L. Lovasz, and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.
- [37] Z. Manna, and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.
- [38] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
- [39] A. Megretski. Relaxations of quadratic programs in operator theory and system analysis. *Operator Theory: Advances and Applications*,

- v. 129, pp. 365–392. Birkhauser -Verlag, 2001.
- [40] A. Megretski. Positivity of trigonometric polynomials. In *Proc. 42nd IEEE Conference on Decision and Control*, pages 3814–3817, 2003.
 - [41] S. Mitra. *A Verification Framework for Hybrid Systems*. Ph.D. Thesis. Massachusetts Institute of Technology, September 2007.
 - [42] A. Mine. *Weakly Relational Numerical Abstract Domains*. Ph.D. Thesis. École Normale Supérieure, December 2004.
 - [43] A. Mine. Relational abstract domains for detection of floating point run-time errors, European symposium on programming (ESOP), LNCS 2986, pages 3-17, 2004.
 - [44] C. S. R. Murthy, and G. Manimaran. *Resource Management in Real-Time Systems and Networks*. MIT Press, 2001.
 - [45] G. Naumovich, L. A. Clarke, and L. J. Osterweil. Verification of communication protocols using data flow analysis. In *Proc. 4-th ACM SIGSOFT Symposium on the Foundation of Software Engineering*, pages 93–105, 1996.
 - [46] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.
 - [47] Y.E. Nesterov, H. Wolkowicz, and Y. Ye. Semidefinite programming relaxations of nonconvex quadratic optimization. In *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Dordrecht, Kluwer Academic Press, pp. 361–419, 2000.
 - [48] F. Nielson, H. Nielson, and C. Hank. *Principles of Program Analysis*. Springer, 2004.
 - [49] P. A. Parrilo. Minimizing polynomial functions. In *Algorithmic and Quantitative Real Algebraic Geometry*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, v. 60, pp. 83-100, 2003.
 - [50] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. Ph.D. Thesis, California Institute of Technology, 2000.
 - [51] D. A. Peled. *Software Reliability Methods*. Springer-Verlag, 2001.
 - [52] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
 - [53] S. Prajna. *Optimization-Based Methods for Nonlinear and Hybrid Systems Verification*. Ph.D. Thesis, California Institute of Technology, 2005.
 - [54] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo, SOSTOOLS: Sum of squares optimization toolbox for MATLAB, 2004. <http://www.mit.edu/~parrilo/sostools>.
 - [55] S. Prajna, and A. Rantzer, Convex programs for temporal verification of nonlinear dynamical systems, *SIAM Journal on Control and Opt.*, 46(3):999–1021, 2007.
 - [56] M. Roozbehani, A. Megretski, E. Feron. Convex optimization proves software correctness. In *Proc. American Control Conference*, pages 1395–1400, 2005.
 - [57] M. Roozbehani, E. Feron, and A. Megretski. Modeling, optimization and computation for software verification. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, v. 3414, pp. 606–622, Springer-Verlag 2005.
 - [58] M. Roozbehani, A. Megretski, E. Feron. Safety verification of iterative algorithms over polynomial vector fields. In *Proc. 45th IEEE Conference on Decision and Control*, pages 6061–6067, 2006.
 - [59] M. Roozbehani, A. Megretski, E. Frazzoli, and E. Feron. Distributed Lyapunov functions in analysis of graph models of software. In

Hybrid Systems: Computation and Control, Springer LNCS 4981, pp 443–456, 2008.

- [60] H. D. Sherali, and W. P. Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Applied Mathematics*, 52(1):83–106, 1994.
- [61] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. URL: <http://sedumi.mcmaster.ca>
- [62] A. Tiwari, and G. Khanna. Series of abstractions for hybrid automata. In *Hybrid Systems: Computation and Control*, LNCS, v. 2289, pp. 465–478. Springer, 2002.
- [63] L. Vandenberghe, and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.
- [64] V. A. Yakubovic. S-procedure in nonlinear control theory. *Vestnik Leningrad University*, 4(1):73–93, 1977.
- [65] <http://web.mit.edu/mardavij/www/Software>
- [66] <http://www.astree.ens.fr/>